DESIGN AND IMPLEMENTATION OF A GRAPHICS PACKAGE FOR COMPUTER AIDED DESIGN OF LSI/VLSI SYSTEMS LAYOUTS USING CIF.

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

By

M. M. AHMAD

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

APRIL, 1985

CERTIFICATE

Certified that the thesis entitled 'DESIGN AND IMPLEMENTATION OF A GRAPHICS PACKAGE FOR COMPUTER AIDED DESIGN OF LSI/VLSI SYSTEMS LAYOUTS USING CIF' has been carried out by Shri M.M. Ahmad under my supervision and has not been submitted elsewhere for the award of a degree.

April, 1985

(Dr. R. Raghuram)
Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology
KANPUR 208016

EE-1985-M-AHM-DES

In loving memory of

my mother

Mrs. Rabia Khatoon

ACKNOWLEDGEMENT

I wish to express my heartfelt thanks to

- My thesis supervisor, Dr. R. Raghuram for suggesting me this current topic and for his inspiring guidance, constant encouragement and active involvement during the course of this work.
- Dr. M.M. Hasan for his invaluable suggestions and for helping me from time to time by providing relevant literatures.
- Dr. E. Bhagiratha Rao, Director, DLRL, Hyderabad for sponsoring me to the M.Tech. program and my senior officer Brig. B.Y. Shankar Narayan for providing me the opportunity to do this course.
- All my friends and colleagues especially, Mr. P.A. Hussain and Mr. K. Siva Krishna Reddy, for their timely help and making my stay at IIT Kanpur enjoyable and memorable.
- My wife who displayed a remarkable sense of resilence by keeping me away from the household chores.
- Mr. J.S. Rawat for his excellent typing of this thesis.

Kanpur
April, 1985

- M.M. AHMAD

ABSTRACT

A graphics package,GICSYS, for computer aided design of LSI/VLSI systems' layouts using Caltech Intermediate Form (CIF), has been designed and implemented on DEC-1090 system as the host machine. It has been written in PASCAL. The graphics display used for this package is the Tektronix 4006-1 a DVST type terminal.

The package consists of an interpreter program and graphics routines for drawing stipple codes (patterns) for different kind of important layers of the integrated systems layouts. The interpreter interprets the commands of CIF and initiates semantic actions through the graphics routines. Some of the important systems layouts have been displayed/ demonstrated using this package. The complete package has been written in PASCAL, a high-level machine independent language. The package is device independent.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 AN IMPLEMENTATION PROCESS OF AN INTEGRATED SYSTEM:

A system designer of LSI/VLSI systems needs certain design tools and procedures for implementing integrated system designs. Various design tools are available that help the designer to produce the geometrical layout patterns for each layer of an integrated system, given the logic, circuit, or topological level design of the system. Similarly, various procedures are also available for encoding the layout patterns. The encoded layouts are subsequently used in the patterning and fabrication processes to implement the integrated system. Some of these procedures are performed manually while others are machine-assisted. To have an appreciation of the work undertaken by us, it is worth summarising the procedures and artifacts of hand layout, and layout description and digitization using a layout language with the help of Fig. 1.1.

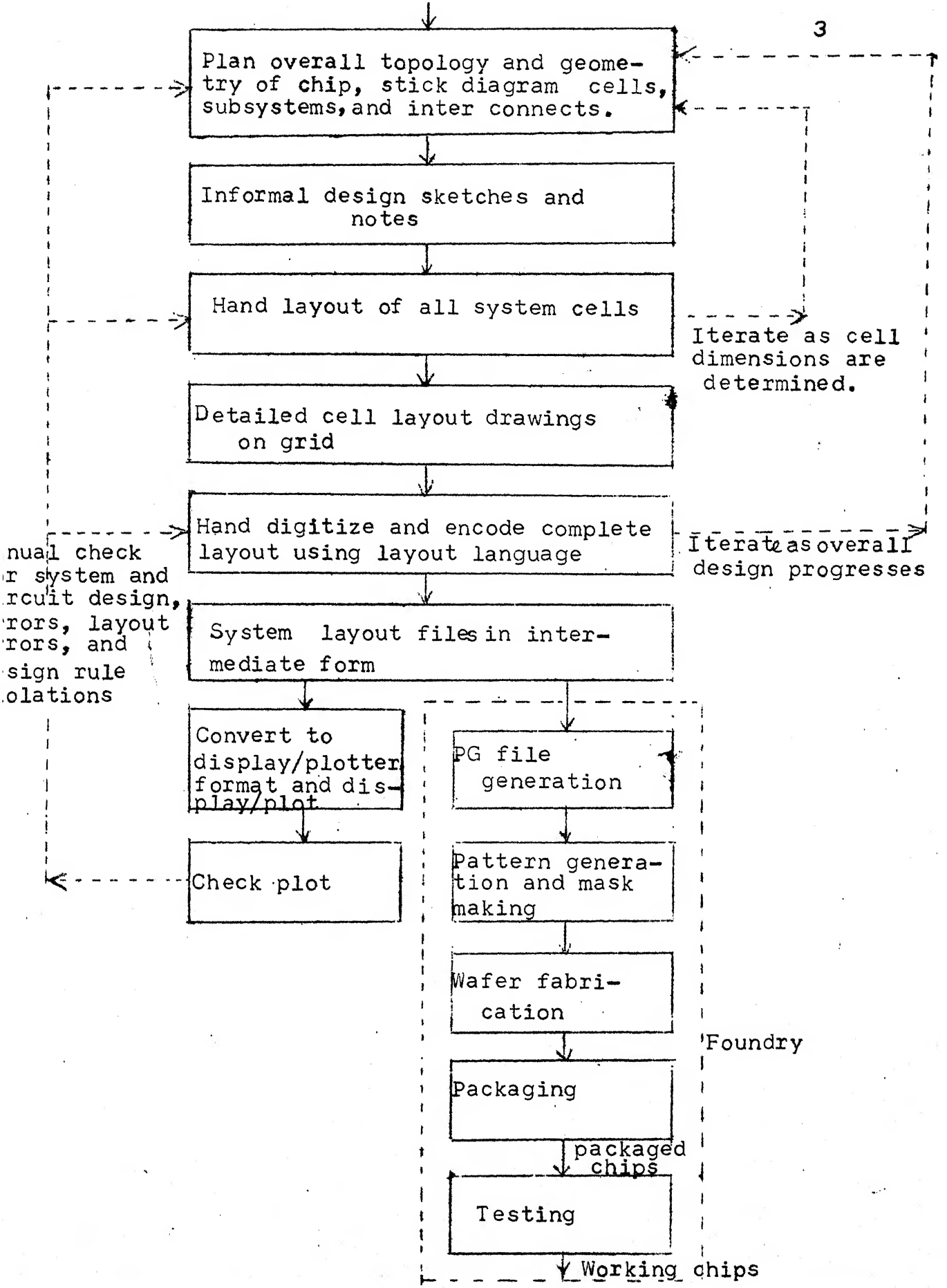
The figure gives a complete, though primitive, sequence of steps to prepare a design for implementation. These procedures are entirely adequate for organizating and

implementing moderate sized LSI chips. The procedures may also be used for large integrated systems which have highly structured design. Our work has direct relevance to the procedures shown in the Fig. 1.1 (except the procedures related to foundry).

1.2 OBJECT AND SCOPE OF PRESENT WORK:

The aim of the project is to obtain check plots of cells/systems layouts. The computer generated layout is first displayed on a graphics terminal and then the check-plots are obtained in the form of photographed plates. Once the final check plot has been obtained, the resulting system layout files in intermediate form (usually in CIF, as in our case) can be sent to foundry for pattern generation, mask making, and wafer fabrication resulting in working packaged chips.

Keeping the above objective in view, we developed this graphics package to display the layouts of integrated subsystems/systems on a graphics terminal, namely, Tektronix 4006-1, Direct View Storage Tube (DVST) terminal. The layout has to be described in a language called Caltech Intermediate Form (CIF) whose details are given in Chapter 2, and this description is fed as input to the program. If the layout description is available in some language other than CIF



g. 1.1: Design, hand layout, design file generation, and design checking using a layout language and simple machine

it has to be translated to an intermediate form, usually to CIF. For example, if the layout has been hand drawn , which is a simple and common method of producing system layouts, it has to be digitized by encoding it in a layout language, say, in CIF. Though we have chosen the CIF form for layout description, there are two other commonly known representations for layout description. These representations are 'symbolic layout language' or 'an interactive design program in any higher level languages'. Even if any of these representations is chosen, it has to be ultimately translated to an intermediate form, usually to CIF which is a standard form being used almost every where. Implementation of interactive computer program is not presently feasible here due to lack of interactive display facilities. One could, however, choose the symbolic layout language which is more user friendly and whose function, in its simplest form, is similar to that of a macroassembler.

The package has been designed, written in PASCAL, a high level machine independent language. It has been implemented on the DEC-1090 system using the Tektronix -DVST terminal. All the important graphics routines have been designed and written by us while a few of the built-in routines of GPGS have been utilized in this package. (GPGS is a set of

subroutines coded in Assembly Language of the host machine, (DEC-1090 system) and once the high-level language interface software module is built, then GPGS is available to the user in the form of library of graphical functions and procedures. GPGS is device independent.) The GPGS routines have been accessed through PASCAL.

Error reporting/handling is one of the very important aspects of any software system. With a view to report errors at lexical and syntactic level, a parsing table was designed and constructed whose details are given in Chapter 3.

The heart of the project is design and implementation of an interpreter program which interprets the statements/commands of CIF file and initiates semantic actions by calling appropriate graphics routines in the program. Its details are given in Chapter 4.

Some illustrative systemslayouts have been given in Chapter 5. These layouts were displayed on the graphics terminal and check plots of these layouts were obtained in the form of photographed plates. These plates have been shown therein, i.e. in Chapter 5.

CHAPTER 2

THE LANGUAGE FOR LAYOUT DESCRIPTION

The Caltech Intermediate Form (CIF Version 2.0) has been chosen as the language for layout description. (The reader is referred to Chapter 4 of Ref. [1]). It is a means of describing graphic items of interest to LSI circuit and system designers. Its purpose is to serve as a standard machine readable representation from which other forms can be constructed for specific output devices viz. display terminals,plotters etc. The form is a fairly readable text file.

The basic purpose of the form is to specify literally every geometric object in the design using ample precision. The features of the form are:

a) It enables sharing designs with others.

b) It allows combining several layout files to form a larger chip.

No matter what the original input methods are (hand layout and coding or an interactive design program),the designs will be translated to CIF as an intermediate step, before being translated again to a variety of formats for

output devices (viz. display terminals or plotters) or other design aids.

2.1 SYNTAX:

The CIF file contains a list of commands followed by an end-marker. The commands are separated with semicolons. The syntax is formally defined as follows. The standard notation proposed by Niklaus Worth (ref. [2]) has been used.

a) = is used to relate identifiers to expression,

b) | is used for OR.

c) " " around terminal,

d) { } indicate repetition any number of times including zero.

e) [ ] optional factor (i.e. zero or one repetition)

f) ( ) are used for grouping.

g) . rules are terminated by it(called 'period').

Syntax allows blanks before and after commands. The syntax reflects the fact that symbol definitions may not nest.

cifFile = {{blank} [command]semi} endCommand{blank}.

command = primCommand|defDeleteCommand|
defStart Command semi{{blank}[primCommand]
semi}defFinishCommand.

```
primCommand              = polygonCommand|boxCommand|roundFlash
                           Command|wireCommand| layerCommand|call
                           Command|userExtensionCommand|
                           CommentCommand.
polygonCommand           = ''P'' path .
boxCommand               = ''B'' integer sep integer sep point
                           [sep point].
roundFlashCommand        = ''R'' integer sep point.
wireCommand              = ''W'' integer sep path.
layerCommand             = ''L''{blank} shortname.
defStartCommand          = ''D''{blank} ''S'' integer [sep integer
                           sep integer].
defFinishCommand         = ''D''{blank} ''F''.
defDeleteCommand         = ''D''{blank} ''D'' integer.
callCommand              = ''C'' integer transformation.
userExtensionCommand     = digit userText.
commentCommand           = ''('' comment Text'')''.
endCommand               = ''E''.
Transformation           = {{blank} (''T'' point|''M''{blank}''X'' |
                           ''M''{blank}''Y''|''R'' point)}.
path                     = point{sep point}.
point                    = sInteger sep sInteger.
sInteger                 = {sep}[''-''] integerD.
integer                  = {sep}integerD.
integerD                 = digit{digit}.
```

```
shortname       = c[c][c][c].
c               = digit|upperChar.
userText        = {userChar}.
commentText     = {commentChar} |commentText ''(''
                  commentText'')''CommentText .
semi            = {blank} '';'' {blank}.
sep             = upperChar|blank.
digit           = ''0''|''1''|''2''|''3''|''4''|''5''|
                  ''6''|''7''|''8''|''9'' .
upperChar       = ''A''|''B''|''C''|...|''Z'' .
blank           = any ASCII character except digit,
                  upperChar, ''-'',''('','')'',or '';''.
userChar        = any ASCII character except '';'' .
commentChar     = any ASCII character except ''('' or '')''
```

2.2 SEMANTICS:

It is important that all readers and writers of files in this form have exactly the same understanding of how the file is to be interpreted. Floating point numbers are not used in CIF . and there are no iterative constructs. There may be additions to CIF in future.

There is provision for defining and calling symbols; this results in substantial reduction in the size of the CIF file.

It is important that the programs, processing CIF files are correct and errorfree.

2.2.1 Measurements:

CIF uses a right handed coordinate system. (Directions and distances are always interpreted in terms of the front surface of the chip.) Distance measurement units are hundredths of a micron (μm). There is practically no limit on the size of a number, but care should be taken to ensure that overflow does not occur.

2.2.2 Direction:

Angle of rotation is not measured/specified directly in radians or degrees. CIF uses a pair of signed/unsigned integers to specify a 'direction vector' which is a measure of angle of rotation. This feature eliminates the needs of trigonometric functions in many situations and avoids the problem of choosing units of angular measurements. The Ist component of direction vector is along +ive X-axis while the other is along +ive Y-axis. Thus a direction vector pointing along X-axis will always have its second component as zero while the other component can be any +ive integer. A (0,0) vector could also mean a direction vector along X-axis but a warning message should be given because Arctan $(\frac{0}{0})$ is undefined.

In the procedure ROTAT in our program, the direction vector (a,b) has been used to compute the COS and SIN components of the angle of rotation 'θ' as follows:

$$\cos(\theta) = a/\sqrt{(a^2+b^2)}$$

$$\sin\theta = b/\sqrt{(a^2+b^2)}$$

taking care that a and b are not simultaneously 0.

2.2.3 Geometric Primitives:

CIF specifies geometric primitives that specify the two dimensional geometric objects. At present it includes boxes, polygons, flashes and wires. It may be extended to accomodate more exotic geometries. It is not necessary to use a primitive just because it is provided. Blanks and upper case characters are acceptable separators.

We have chosen only blanks as acceptable separators for implementation of our program.

2.2.3.1 Boxes:

Box Length 10 Width 20 Centre 40, 30
Direction 1,1; (or B 10 20 40 30 1 1;)

Box is the most widely used geometric primitive for the layout description in CIF. It is specified by the boxCommand.

The first character of the command is 'B' which denotes that it is a box. The Ist argument of the command is length, the second is width, 3rd and 4th arguments specify the X and Y coordinates of the centre of the box respectively, 5th and 6th arguments specify the direction vector which represents the orientation of the box. Length is the dimension of the box parallel to the direction vector while Width is the dimension perpendicular to the direction vector. The fields that define a box are shown in Fig. 2.1.

2.2.3.2 Polygons:

Polygon A 0,0 B 5,10 C 10,15 D 15, 20;(or P 0 0 5 10 10 15 15 20;)

A polygon is an enclosed region specified/determined by the vertices given in the path, in order (see Fig. 2.2). For a polygon having n sides, n vertices are specified and the last vertex is joined to the first vertex. Polygon Command starts with 'P' and the subsequent pairs of integers/signed-integers stand for X,Y coordinate-specifications of vertices.

The programs that try to interpret polygons may place several restrictions on their paths. No set of constraints has been found generally acceptable and likewise no program currently exists for dealing with completely general polygon.

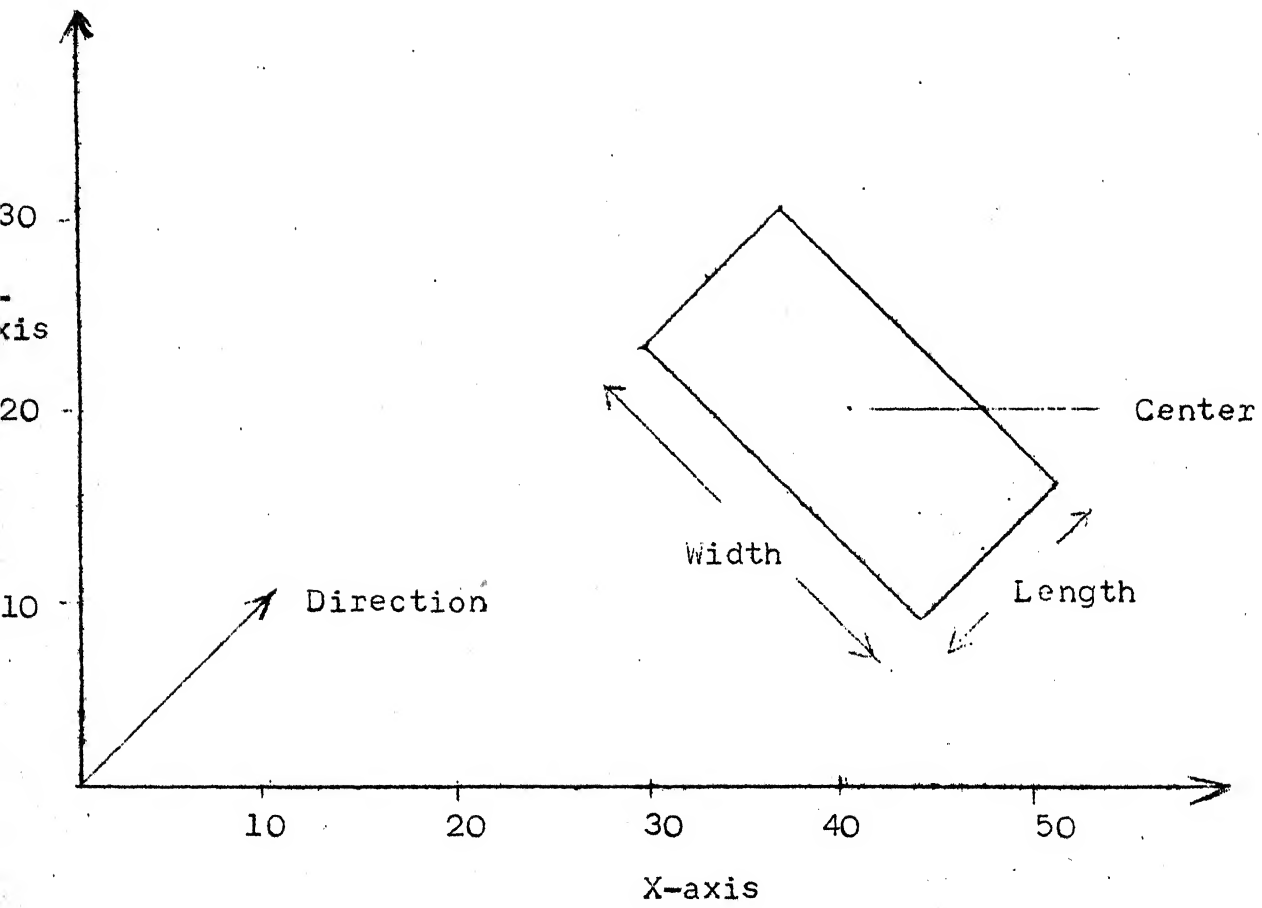


Fig. 2.1: Box representation in intermediate form.

No subroutines have been developed by us for polygon-shaped portion of the layout. All the illustrative examples, given by us in Chapters 4 and 5, are based on the 'boxes' only as the primitive geometric objects.

2.2.3.3 Flashes:

RoundFlash Dia 20 Centre 50, 30; (or R 20 50 30 ;)

These are specified by the RoundFlashCommand. The first character of the command is R. The Ist argument of the command which is an unsigned integer, specifies the diameter. The 2nd and 3rd arguments specify the coordinates of centre of the flash which is circular in shape. (see Fig. 2.2).

2.2.3.4 Wires:

Wire Width 40 A 0,0 B 20, 10 C 50 60 (or W 40 0 0 20 10 50 60;)

It is specified by the wire command which starts with 'W'. Its arguments, in pairs, specify the points along its path. It is sometimes convenient to describe a long uniform width traversed by the path along its centre line. We call this construct a wire (see Fig. 2.2). An ideal wire is one which includes a semicircular cap on both ends of each of its segments. The semicircular cap, ensures a smooth connection between segments in a wire and between touching wires.

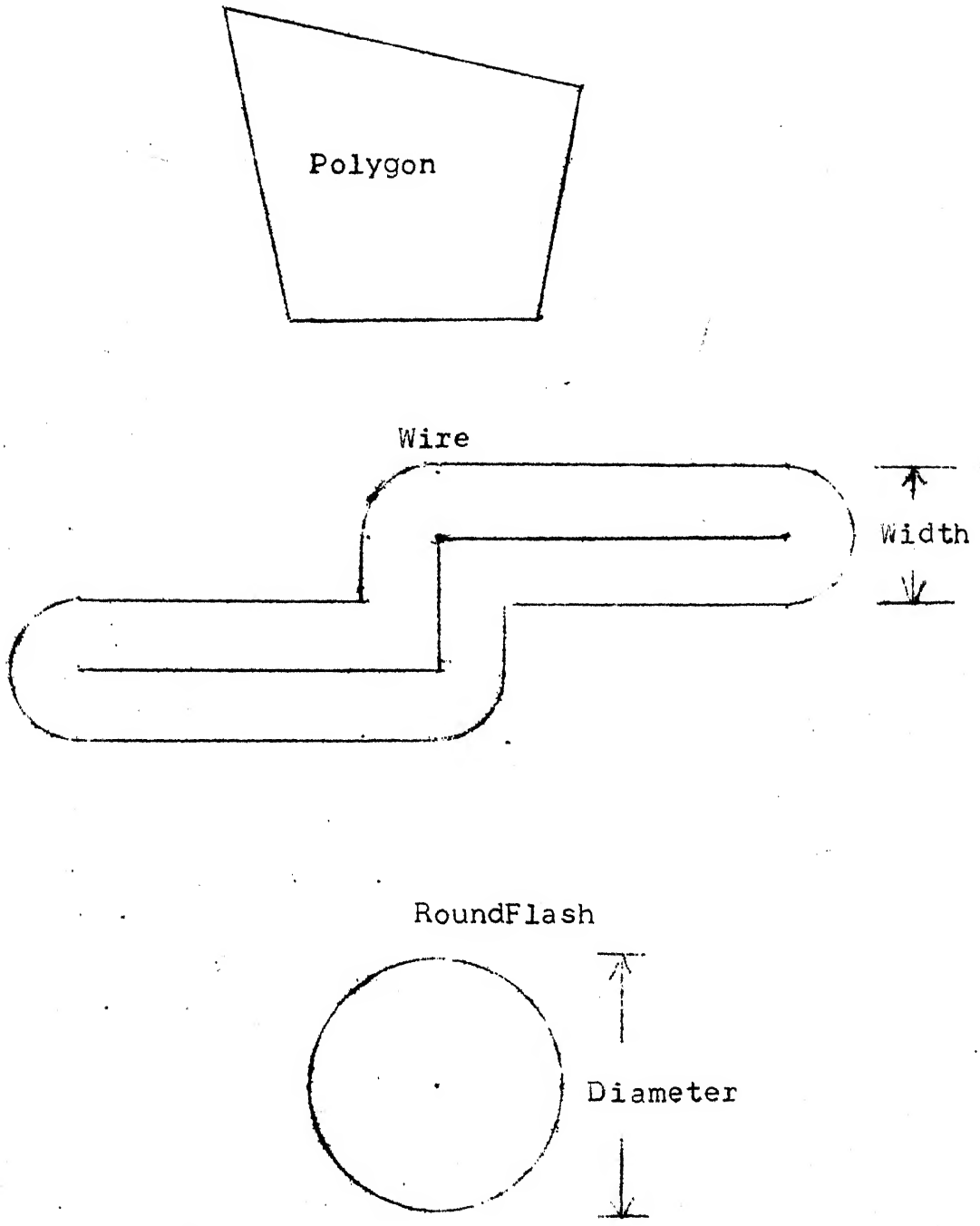


Fig. 2.2: Other items in the intermediate form.

For output devices, which take appreciably longer time to construct circles, the ideal wire is approximated with squared off ends. It may be noticed that squared off ends work nicely for segments meeting at right angles but cause problems if wires or wire segments are connected at arbitrary angles. To overcome this problem, any wires can be converted into connected sets of boxes of appropriate dimensions. The width of each box is same as width of wire. However, the length of the boxes must be adjusted to minimise unfilled wedges and overlapping ears. An algorithm for constructing boxes from a wire description is given in Section 4.5.4 of Chapter 4 of ref.[1].

2.2.3.5 Layer Specification:

Layer must be specified for each primitive geometry element. Layer is specified as a 'mode' that applies to all subsequent primitives, until the layer is set again. Layer mode is preserved across symbol calls. Layer is specified by the layer command. The 1st character of the command is the letter 'L' denoting that this is a layer specification. The intention of the layer command is to label locally the layer for a particular geometry. It is, therefore, senseless to specify any geometry without any layer specification.

To avoid conflict at the time of combining several files in intermediate form, it is very important that layer names must be unique. The basic idea is that the first character after L, denotes the technology and the remainder is mnemonic for the layer.

We have chosen the following layers, which are more commonly used, for our purpose:

ND	nMOS Diffusion
NP	nMOS Polysilicon
NC	nMOS Contact Cut
NM	nMOS Metal
NI	nMOS depletion mode implant
(NZ	is a dummy layer)

New layer names can be defined as and when needed.

2.2.4 Symbols:

Most of the LSI/VLSI layouts include items that are often repeated. Because of this repetition it is helpful to define often used items as 'Symbols'. This facility, together with the ability to 'call' for an instance of the symbol to be generated at a specific position, greatly reduces the size of CIF file. Thus this feature of defining and calling symbols is a very important aspect of CIF files. This feature is similar to defining and calling picture segments in the

graphics terminology or similar to defining and calling sub-circuits, in the network terminology, to draw/simulate the complete circuit.

The symbol facilities are deliberately limited to avoid undue complexity of implementing programs that process CIF files. For example:-

a) Symbols have no parameters.

b) Symbol geometry is not scaled up or down while calling the symbol.

c) There are no direct iterative constructs.

The main reason for symbol facilities is to limit the file size. But if the symbol mechanism is not adequate for some applications, the desired geometry can be still be achieved with lesser use of symbols and more use of explicit geometrical primitives. Symbols need not be used at all, this avoids the necessity for intermediate storage for symbol definitions; but results in larger CIF file. Our program covers both these aspects.

The ability to call for iterations (arrays) of symbols is not provided in CIF. This is primarily because of the difficulty of defining a standard method of specifying iterations. One can still achieve a great deal of file compaction by defining several layers of symbols (e.g. cell,

row, double row, array). Implementation of iterations is the future prospect for addition to CIF.

2.2.4.1 Defining Symbols:

Definition Start No.40 A/B = 200/1; ...;
Definition Finish ; (or DS 40 200 1; ...; DF;)

A symbol is defined with DS as its first command and DF as its last command. The first argument of DS command is an identifying symbol No..Symbol No. is unrelated to the order of listing of symbol definitions in the file.

The second and third arguments of DS command which are optional parameters, are meant to scale distance measurements. We will call the 2nd and 3rd argument as 'a' and 'b' respectively. As CIF is read, each distance (size or position) measurement, mentioned in the geometric primitive commands (Boxes etc.) and call commands in the symbol definition is scaled to

(a * distance)/b

For example, if the designer uses a grid of 1 micron, the symbol definition might specify all distances in microns and specify a = 100 and b = 1 or the designer might choose λ (characteristic fabrication dimension) as a convenient unit. This provision reduces the number of characters in CIF file

by shrinking the integers that denote dimensions and may improve the legibility of the file. However, it should be clear in mind that it does not provide scaling or ability to change the size of a symbol called within the symbol-definition.

Nesting of definitions is not allowed. That is, after a DS command is specified for a particular symbol, the terminating DF must precede the next DS. The definition, may, however, contain calls to other symbols which may in turn call other symbols.

A symbol must be defined before it is called by the symbolcallCommand. This can be achieved by placing all definitions first in the file and then placing the call commands.

When several people contribute to a design, some symbol management is necessary. For example, if a symbol is redefined, the previous definition is discarded. This is achieved by defDeleteCommand given below.

2.2.4.2 Calling Symbols:

Call Symbol No. 40, Mirrored in X, Rotated to 1 1, Translated to 5 15; (or C 40 MX R 1 1 T 5 15;)

The call command is used to call a specified symbol and to specify transformation that should be applied to all

geometry contained in the symbol definition. The call command starts with letter 'C'. The call command recognizes the symbol to be called with its 'symbol index' established at the time of defining the symbol.

The transformation to be applied to the symbol is specified by a sequence of primitive transformations mentioned in the 'Call' command. These primitive transformations are:

T point	Translate the current symbol origin to this point.
MX	Mirror in x i.e. multiply x-coordinate by -1.
MY	Mirror in y i.e. multiply y-coordinate by -1.
R point→Rotate	symbol's X-axis to this direction.

Ordering of these primitive transformations is very important. These primitive transformations are applied in sequence i.e. first primitive transformation given in the call command will be applied first, then the second one and so on. It is not necessary to apply each primitive transformation separately; instead, the several primitive transformations can be combined into one matrix multiplication.

Symbol calls may nest i.e. a symbol definition may contain a call command which calls another symbol. When calls nest, it is necessary to 'concatenate' the effects of the transformations specified in the various calls (see

section 2.3). A CIF file comprising of a nested call is appended below. In this, call command C 15 T 30 0; calls symbol No. 15 which in turn calls symbol No. 12 vide the call command C 12 T 20 20.

Layer settings are preserved across symbol calls and definitions.

```
LNZ;
DS 12;
LND;
B 4 4 5 2 1 0 ;
B 4 4 5 21 1 0 ;
LNI;
B 5 10 5 15 1 0;
DF;
DS 15 ;
LNI;
B 20 10 -8 -4 20 -20;
C 12 T 20 20;
B 30 30 10 4 1 0;
DF;
C 12 T 10 -10 R 1 0;
C 15 T 30 0 ;
E
```

2.2.4.3 Deleting Symbol Definitions:

The DD command tells that all symbols with indices greater than or equal to the number specified as the argument of the command can be 'forgotten' as they will not be called again (e.g. DD 50 means that Delete definitions >= 50). This feature is included so that several CIF files can be appended and processed as one. In such a case, it is necessary to delete symbol definitions used in the 1st part of the file because of the following reasons:

a) Definitions may conflict with definitions made later

b) A great deal of storage (memory) can usually be saved by deleting the old definitions.

The provision, that some definitions are to be retained (while others are to be deleted) is aimed at maintaining a standard library of symbols/definitions that a project group may develop. These standard symbols may be pull up transistors, contacts,etc.

At present this feature is not incorporated in our program but can be implemented in future.

2.5 User Extension Command:

Several command formats (any command starting with digit) are reserved by individual users. One of the aims

of this command is to record ancillary information or data structures(e.g. circuit diagrams, design-rule check results) that are to be maintained in parallel with the geometry specified in CIF form. An illustrative example will look like:- 4:NONSTANDARD DESIGN RULES: LAMBDA=4.0 ;

2.2.6 Comments:

Comment facility is provided to make the file easier to read. The comment text is enclosed within a pair of parentheses.

This feature has been implemented by us in our program.

2.2.7 End Command:

This command is meant to indicate end of file. This is the last command of a CIF file. The command consists of just one character i.e. E.

2.3 TRANSFORMATIONS:

When we are calling a symbol by a call command, we need to apply a transformation to the coordinates of an item given in the symbol definition to get it transformed into the coordinate system of the chip.

Since we allow no scaling in a symbol call, distances are never changed by the symbol calls. Thus a distance requires no transformation.

A point (x,y) given in the symbol is transformed to a point (x',y') in the chip coordinate system by a 3x3 transformation matrix T:

$$[x' \; y' \; 1] = [x \; y \; 1] \, T$$

We are dealing with homogeneous coordinate system. T itself may be the product of primitive transformation matrices T1, T2, T3 etc.

$$T = T1 \; T2 \; T3$$

where T1 is a primitive transformation matrix obtained from 1st primitive transformation mentioned in the call command, T2 from the 2nd and so on. There may be fewer or more than 3 primitive transformations in a call command.

The primitive transformation, matrices are given below:-

T_{xy} $\qquad T1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & y & 1 \end{bmatrix}$

MX $\qquad T2 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$$MY \quad T3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_{xy} \quad T4 = \begin{bmatrix} x/c & y/c & 0 \\ -y/c & x/c & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{where } c = \sqrt{(x^2+y^2)}$$

[Note: $x/c=\cos\theta$ and $y/c = \sin\theta$ if θ is the angle of rotation measured in counter-clockwise direction].

Nested calls require that we combine the transformations already in effect with those specified in the new call. Suppose we are calling a symbol a, transforming it to coordinates of chip by the matrix T_{ac} and under a's definition there is call to symbol b. The transformations specified in the call will result in a matrix T_{ba} that will transform coordinates specified in b to the coordinate system used in a. Now these must be transformed by T_{ac} to convert from the system of symbol a to that of the chip. Thus full transformation becomes

$$[x' \; y' \; 1] = [x \; y \; 1] \; T_{ba} \; T_{ac}$$
$$= [x \; y \; 1] \; T_{bc}$$

Thus only one transformation T_{bc} can be applied to convert directly from the coordinates in symbol b to the chip. This procedure can be carried out to any arbitrary level of nesting.

To implement transformations, one can proceed as follows: A 'current transformation matrix' CT can be maintained, which is initialised to the identity matrix. This matrix CT is used, then, to transform all coordinates. When a symbol call is encountered, following steps are carried out:

1. 'Push' the current transformation and layer name on stack.
2. Set layer name to ZZZZ
3. Collect the individual primitive transformations specified in the call into the matrices T1,T2,T3 etc.
4. Replace the current transformation CT with T1 T2 T3... CT (i.e. premultiply the existing transformation by the new primitive transformations, in order).
5. Now process the symbol using the new CT matrix.
6. When the symbol call is completed, 'pop' the saved matrix and layer name from the stack. This restores the transformation to its state immediately before the call.

We have implemented the nested calls, based on an algorithm similar to that mentioned above. The procedures PUSHTRN, POPTRN, PREMULTIPY, IDENT, IDENCT, BEGINTRN have been designed and written by us to implement it. At present, the program caters for five levels of nesting but it can be made to cater for any arbitrary level of nesting by simply changing the value of the constant STACKSIZE by the formula

STACKSIZE = 6 * Max level of nesting.

CHAPTER 3

THE DESIGN AND IMPLEMENTATION OF LEXICAL ANALYSER AND PARSING TABLE

We know that error reporting/handling is a very important aspect of any software package. To detect errors at syntactic level, design and implementation of a parser is imperative. But design of a parser involves the design and implementation of a lexical analyser. The lexical analyser which is the 1st phase of any computation process, separates the characters of the source language (CIF, in our case) into groups that logically belong together, called tokens, The output of the lexical analyser is stream of tokens, which is passed to next phase i.e. to the parser. A parser for any grammar G is a program that takes as input a string w and produces as output either parse tree for , if is a sentence of G or an error message indicating that w is not a sentence of G.

3.1 DESIGN AND IMPLEMENTATION OF LEXICAL ANALYSER:

A lexical analyser was designed for the simplified version/subset of the original grammar of CIF. The simplified version/subset of the original grammar is given at Appendix A. It contains 40 terminal symbols. Each of these

terminal symbols consists of just a single character.

Data structure: The data structure chosen was arrays and arrays of records.

a) SYMBOL is a record having two fields i.e. SYMNAME and SYMCODE.

   SYMNAME is for symbol name and symcode represents the code for corresponding symbol name.

b) SYMTAB is an array which stores the symbol information i.e. symbol name and its corresponding code. This array is basically a symbol table.

Description of Procedures:

i) Procedure INSERT: This procedure inserts a valid symbol in symbol table (Symtab). The entries of the symbol table are symbol name and its code.

ii) Procedure NEXTTOKEN.: checks whether a character, read, is a valid symbol or not. If it is valid symbol, then it inserts it in the symbol table by calling the proc. INSERT else it gives an error message saying 'Illegal symbol found'.

Main Program: It reads the characters of input file one by one and calls the procedure 'NEXTTOKEN . In case

of valid symbols (characters), the symbols are stored in the array SYMTAB otherwise an error message is given. The program is in the file LEX2.PAS.

3.2 CONSTRUCTION OF SLR(1) PARSING TABLE:

The parser checks the input and gives suitable diagnostic messages in case of any syntactic errors in the input.

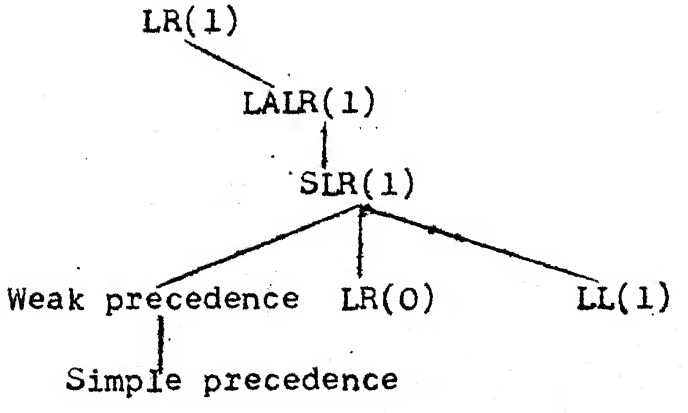
Two of the widely used parsing techniques for context free grammars are TOP-DOWN (LL(1)) and Bottom-Up (LR(1)) parsing. The reader is referred to Chapters 5, 6 of Ref. [3] for a good treatment for both of the above techniques. Of the above two, LR is more powerful as it accepts a larger class of grammars. There are three different techniques for producing LR parsing table:

i) LR(1)

ii) LALR(1)

iii) SLR(1)

The grammar hierarchy is shown below:

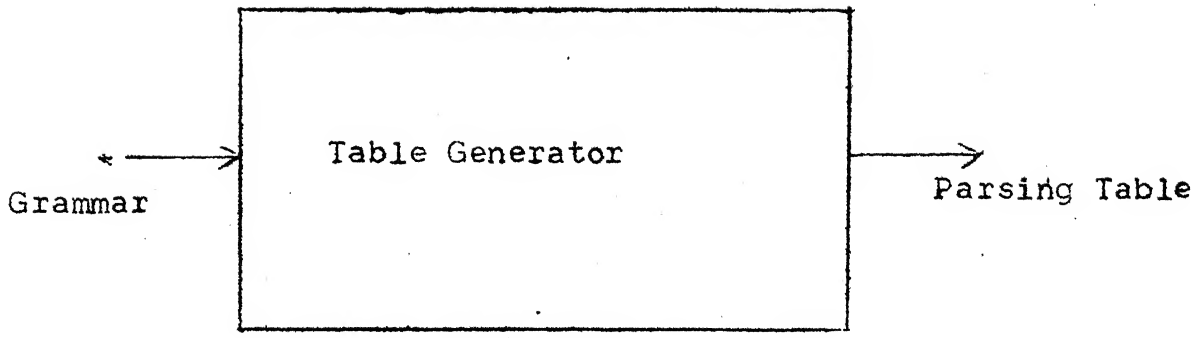


Out of the three LR techniques [LR(1), LALR(1), SLR(1)], SLR(1) is simplest to implement but it is less powerful compared to the other two. LR(1), also called canonical LR, is the most powerful and will work on a very large class of grammars; but it is most difficult to implement. LALR(1) is intermediate in power between SLR(1) and LR(1). The size of the parsing table (i.e. number of states) is same whether we use SLR(1) or LALR(1) algorithm. It is only LR(1) algorithm which produces a larger table.

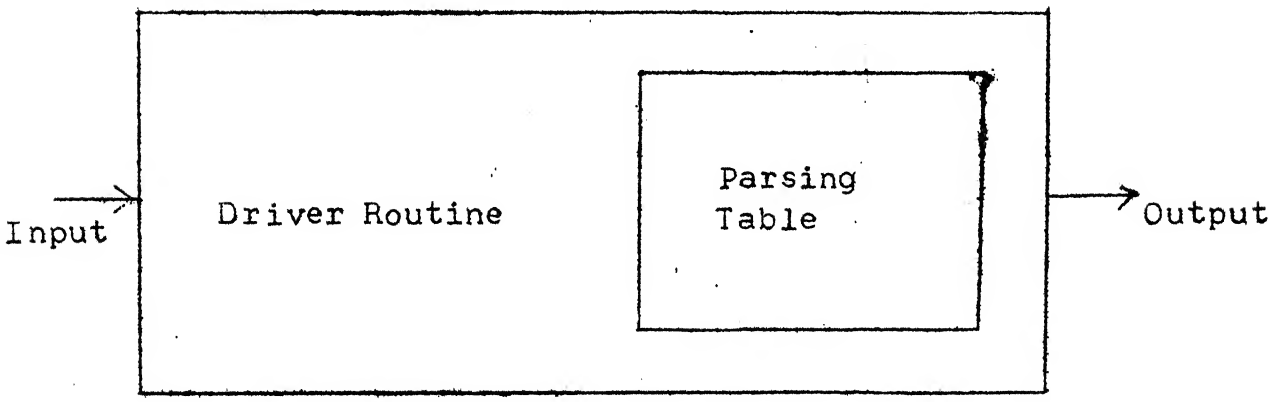
Since, in practice, most features of programming languages turn out to be SLR(1) and it is easiest to implement, we have chosen SLR(1) algorithm for construction of our parsing table.

An SLR(1) parser consists of two parts, a driver routine and a parsing table. The driver routine is same for all SLR(1) parsers, only parsing table changes from one parser to other. Thus generating the parser requires generating the parsing table from the input grammar and then integrating the parsing table with the driver routine. This is schematically shown in Fig. 3.1.

The SLR(1) parsing table was constructed by hand by us as the grammar was not too large. If available, the automatic parser generator is the best tool for large grammars.



(a) Generating the parser



(b) Operation of the parser

Fig. 3.1: Generating an LR parser.

The rectangular parsing tables, like ours, are fast to access and provide good diagnostic facilities; but they take up too much storage space. The techniques of sparse matrices or other techniques can be applied to achieve increased space-efficiency; but,then time taken to access a particular element of the table increases or it causes delayed detection of syntax errors. By delayed detection, we mean that more parser actions have taken place before an error is recognized.

3.2.1 Representation of Input Grammar in BNF Form:

The input grammar i.e. syntax of CIF,in its original form,was not suitable for the purpose of the design of the parser and therefore it needed its conversion into equivalent BNF representation. The original form contained repetitions denoted by curly brackets: this representation has been converted to a recursive relation. For example, A = { B } has been converted to a recursive relation

A :: = e('e' being the null/empty string)

A :: = AB

The square brackets have been replaced after converting the relevant production into two equivalent productions e.g. [a] has been replaced as follows. If the

production is al = [a] then it has been converted to:-

al ::= e

al ::= a

Since we had to design the parser by hand, we simplified the original grammar by retaining only the salient features of the grammar. Restrictions were placed on the number of 'blanks' to be allowed. Certain commandsviz. Polygon Command, Wire Command, Round Flash Command, User Extension Command, Comment Commands were deleted. Blanks are the only acceptable separators, etc.

This simplified version of the grammar was then taken as the input grammar for designing the parser. This is given at Appendix A. To distinguish non-terminals from terminals, pairs of horny brackets have been put around non-terminals.

3.2.2 Computation of Functions:

We need certain functions to enable us to proceed ahead to design a parser. These functions are: (a) FIRST and (b) FOLLOW (refer to ref.[3]).

a) FIRST - If α is any string of grammar symbols, then FIRST (α) is the set of terminals that begin strings derived from α. If $\alpha \overset{*}{\Rightarrow} e$, then e is also in FIRST(α).

To compute FIRST(X) for all grammar symbols X, apply the following rules until no more terminals or e can be added to any FIRST set:-

1. If X is terminal, then FIRST(X) is $\{X\}$.
2. If X is non-terminal and $X \rightarrow a\alpha$, then add a to FIRST(X). If $X \rightarrow e$ is a production, then add e to first (X).
3. To compute FIRST(X1, X2,...,Xn) where X1,X2,...,Xn is a string of grammar symbols, proceed as follows:

FIRST(X1, X2,...,Xn)

$=\{$ non-e symbols of FIRST(X1),
non-e symbols of FIRST(X2) if e is in FIRST(X1),
non-e symbols of FIRST(X3) if e is in both FIRST(X1) and FIRST(X2), and
so on $\}$

b) FOLLOW: FOLLOW(A), for non-terminal A, is the set of terminals a that can appear immediately to the right of A in some sentential form, i.e. $S \xrightarrow{*} \alpha A a \beta$ for some α and β. If A can be the rightmost symbol in some sentential form, then e is in FOLLOW(A).

To compute FOLLOW(A) for all non-terminals A, apply the following rules until nothing can be added to any FOLLOW set:

1. e is in FOLLOW(S), where S is the start symbol.
2. If there is a production $A \rightarrow \alpha B \beta$, then everything in FIRST(β) but e is in FOLLOW(B). Note that e may still windup in FOLLOW(B) by rule (3).
3. If there is a production $A \rightarrow \alpha B$ or a production $A \rightarrow \alpha B \beta$ where FIRST(β) contains e (i.e. $\beta \overset{*}{\Longrightarrow} e$), then everything in FOLLOW(A) is in FOLLOW(B).

Using the above rules, FIRST and FOLLOW functions were computed by us.

c) CLOSURE: If I is a set of items for a grammar G, then the set of items CLOSURE(I) is constructed from I by the rules:-

1. Every item in I is in CLOSURE(I)
2. If $A \rightarrow \alpha . B \beta$ in in CLOSURE(I)and $B \rightarrow \gamma$ is a production, then add the item $B \rightarrow . \gamma$ to I, if it is not already there.

d) GOTO: Another useful function is GOTO (I,X) where I is a a set of items and X is a grammar symbol.

GOTO (I,X) is defined to be the closure of the set of all items $[A \rightarrow \alpha X . \beta]$ such that $[A \rightarrow \alpha . X \beta]$ is in I.

3.2.3 The Sets-of-Items Construction:

The algorithm to construct C, the canonical collection

of sets of LR(0) items for an augmented grammar G' , is given below in Pascal-like language:-

```
Procedure  ITEMS(G');
 Begin
     C: = { CLOSURE( { S'→.S ) } ;
      Repeat
           for each set of items I in C and each grammar
               symbol X such that GOTO (I,X) is not empty
               and is not in C
               do add GOTO (I,X) to C
      until no more sets of items can be added to C
 end
```

3.2.4 Algorithm for Construction of SLR Parsing Table:

Input C, the canonical collection of sets of items for an augmented grammar G'.

Output If possible, an LR parsing table consisting of a parsing action function ACTION and a goto function GOTO.

Method Let $C = \{ I_0, I_1, \ldots, I_n \}$. The states of the parser are $0,1,2,\ldots,n$; state i being constructed from I_i. The parsing actions for state i are determined as follows:

1. If $[A \rightarrow \alpha.a\beta]$ is in I_i and GOTO $(I_i,a)=I_j$, then set ACTION [i,a] to 'shift j'. Here a is a terminal.
2. If $[A \rightarrow \alpha.]$ is in I_i then set ACTION [i,a] to 'reduce $A \rightarrow \alpha$' for all a in FOLLOW(A).
3. If $[S' \rightarrow S.]$ is in I_i then set ACTION (i,$\not$] to 'accept'.

If any conflicting actions are generated by the above rules, we say the grammar is not SLR(1). The algorithm, then, fails to produce a valid parser in this case.

The goto transitions for state i are constructed using the following rules:

4. If GOTO $(I_i,A) = I_j$, then GOTO [i,A]=j.
5. All entries not defined by by rules (1) through (4) are made 'error'.
6. The initial state of the parser is the one constructed from the set of items containing $[S' \rightarrow .S]$.

The parsing table obtained from above algorithm is called the SLR table for G. The parser using this table is called the SLR parser for G, and a grammar having an SLR parsing table is said to be SLR(1).

3.2.5 Implementation of SLR Algorithm:

The above algorithm was implemented for the simplified grammar of CIF to construct a SLR parsing table. The SLR parsing table is given at Appendix B.

3.2.6 Verification of Correctness of Results:

As the table was hand constructed (i.e. without using an automatic parser generator), we thought that it should be verified for correctness of its entries. Fortunately, a program was available for automatic parser generator.

The results obtained by us were compared with those obtained from the automatic parser generator and the two results tallied to each other.

If one takes, the syntax of CIF in its entirety, the number of states of the parsing table will increase substantially and in that case construction by hand becomes very very difficult and we, therefore, suggest the use of automatic Parser Generator for it.

CHAPTER 4

DESIGN AND IMPLEMENTATION OF AN INTERPRETER

Instead of translating the CIF program into machine code and then executing it, an alternative approach is to translate and execute each intermediate language statement as it is encountered. A program which achieves this task, is called an interpreter. It has the following advantages:

1. It is often more compact than the machine code produced by a compiler.
2. An alteration to a part of a program need not involve recompiling the whole program.

The disadvantage of an interpreter is that programs tends to run relatively slow since intermediate language statements have to be translated each time they are executed, but time penalty may not be too significant depending upon the design of the intermediate language.

Since our language is in intermediate form and the statements of the language comprise of simple constructs, say upper case characters (each character representing a token), signed/unsigned integers and a few special characters, we decided to design and implement an interpreter for semantic actions being dictated by the commands of the CIF language. Design and implementation of the interpreter is the most important feature of our program.

4.1 PICTURE SEGMENTATION AND DATA STRUCTURES:

The structure of CIF file is based on the concept of subpicture structure, wherein, the display file (consisting of instructions to draw the geometrical objects, say, boxes) has been divided into segments ('symbols' in the terminology of CIF language), each segment corresponding to a component of the overall display (layout). One can also associate a set of 'attributes' with each segment. One such attribute is visibility. A visible segment will be displayed but an invisible segment will not be displayed.

The SEGMENT TABLE - There is certain necessary information associated with each segment and one has to find out ways to organise this information. Each segment should be given its own unique name so that we can specify it. If we are to change the visibility of a segment, we must have some way to distinguish that segment from all others. When we refer to a display file segment, we must know which display file instructions belong to it. This may be determined by knowing where the display file instructions for the segment begin and how many of them there are. For each segment, we shall need some way of associating its display file position information and its attributes information with its name. We need to organise this information so that given the segment

name, we can look up or alter its attributes, or given its name, we can interpret the corresponding display file instructions.

In our system this has been done by forming a segment table (represented by the array SEGTAB). We have used the 'array of records' as the data-structure for the segment table. SEGTAB is an Array [1.. NOOFSEGS] of SEGINFO where

```
SEGINFO = record
              SEGNO   : integer
              SEGSTADD: integer
              SEGSIZE : integer
              OFFON   : 0..1 {1 for visibility and
                              0 for invisibility}
end;
```

[SEGNO : is the segment name.

SEGSTADD: stands for SEGMENT-START address, specifying the display file starting locations.

SEGSIZE : gives the size of the segment i.e. no. of instructions in the segment.

OFFON : stands for VISIBILITY .]

The schematic diagram of the segment table is shown in Fig. 4.1.

[Note: The word 'display file' used in this chapter means the part of the CIF file consisting of executable instructions to draw the geometric primitives, say, boxes].

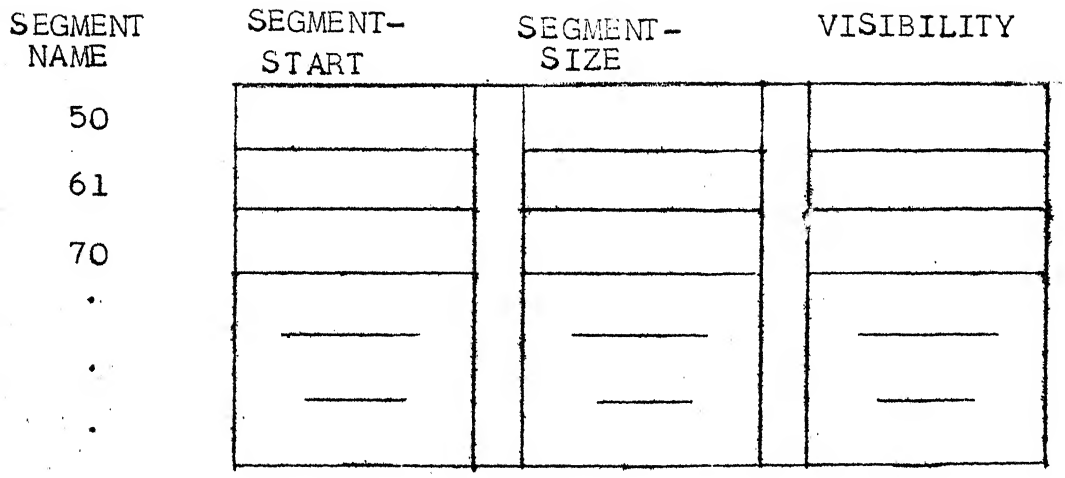


Fig. 4.1: The Segment Table.

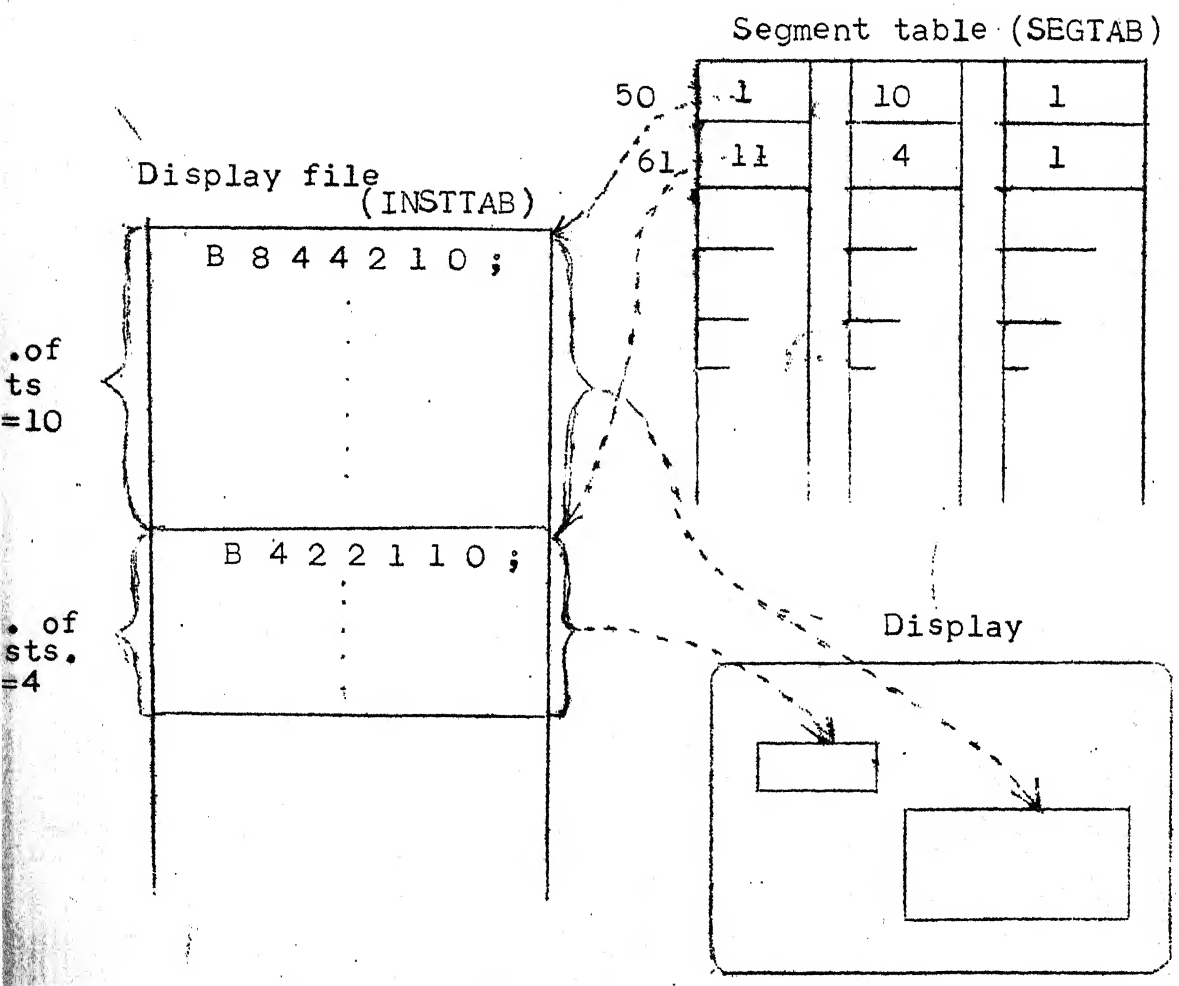


Fig. 4.2: The segment table indicates the portions of the display file used to construct the picture.

For each visible segment, we shall look up its starting point and size and pass this information along to our display file interpreter which will interpret each instruction of the segment.

There are other possible schemes/data structures for implementing the segment table, many with substantial advantage over the one we have chosen. We have selected this design because it allows simple accessing. It does not require any new data-structuring concept, and its updating is straight-forward.

Display File Structure: If the segment is visible and its size is greater than zero, then transformation matrix is applied on it and the segment is interpreted. As mentioned above, the segment table tells where to begin and how many instructions to interpret. We pass this informatio to our interpreter which will interpret each instruction of the segment.

There are many possible data-structures which might be used for necessary operations on the display file. We have chosen 'array of records' as the data-structure. In our system, we have named the display file as a table INSTTAB which is an Array [1..MAXNOOFINST] of INSTREC where

```
INSTREC = record
        INST : Array [1..INSTLENGTH] of char;
        Lay  : 0..5
end;
```

[INST : represents the instruction-contents
Lay : represents the codes for different layers.]

While insertion, selection, and deletions are easy for an array, deletion may not be very efficient. If we wish to remove an instruction at the beginning of the display file, we must move all succeeding instructions. If the display is large, this could mean a lot of processing to recover only a small amount of storage. One alternative data structure which might be used is the linked list. Deletion of cells from a liked list is very easy. In our system, since we have not catered for 'deletions' at present, the array structure is alright.

The role of segment table in conjunction with display file to construct the picture is schematically shown in the Fig. 4.2.

4.2 DESCRIPTION OF PROCEDURES:

a) Procedure MAKETABLES: is a procedure which makes two tables after reading the CIF input file. One of these is SEGTAB and other is INSTTAB as described above.

b) Procedure EXJTHINST: This procedure, when called by the procedure COMMANDPROCESSOR, executes the J_2th instruction of the INSTTAB. If it (i.e. J_2th instruction) is not a call command, the arguments of the instruction are passed on to the corresponding graphics procedure for onward drawing of the primitive geometric object. If it is 'call' command then it initialises the identity matrix and passes the parameters, related to transformation to be applied, to translation matrix/rotation matrix/mirror matrix in the order specified in the call command for onward drawing of the transformed picture-segment specified by the symbol index of the segment.

If the symbol number specified in the Call Command is not there in any of the symbol definitions, then it gives an error message.

c) Procedure COMMANDPROCESSOR: This procedure finds the 1st executable instruction in the symbol definition of the called symbol with the help of SEGTAB which stores the information about starting address i.e. address of the 1st instruction. The SEGTAB also has the information about number of instructions in a particular symbol definition. It calls the procedure EXJTHINST which applies the necessary transformations on the segment and then executes each instruction of the segment sequentially by calling the appropriate graphics routine.

4.3 DESIGN AND IMPLEMENTATION OF THE GRAPHICS ROUTINES:

Five graphics routines (i.e. procedures) were written in PASCAL using two basic routines (functions) of GPGS:- LINE (X,Y,0) and LINE(X,Y). LINE(X,Y,0) is same as MoveTo (x,y) and LINE(X,Y) is same as LineTo (X,Y) functions found in literatures (see ref. [4]). LINE(X,Y,0) sets the current beam position (X,Y) for the next line or text string. LINE(X,Y) draws a straight line from the current beam position to point (X,Y) and resets the starting point to (X,Y). If we want to draw a '.' at point (X,Y) we have to call LINE(X,Y,0) and LINE(X,Y) in succession. Five graphic procedures were designed and implemented to simulate five types of stipple codes (patterns), each code denoting a particular layer type. These procedures are BOXCUT, BOXDIFF, BOXIMPLANT, BOXMETAL and BOXPOLY. Care was taken, while designing these procedures that the pattern behaviour does not shrinks or expands with the changes in the 'BOX' dimensions in the world coordinate system. This was achieved by fixing the number of increment-intervals (denoted by the const NT in the program) in a length of 1.0 of the viewport. Transformation routines have also been designed and written by us.

SELECTION OF WINDOW and VIEWPORT :

For the sake of mere convenience, (XLOW, YLOW) of WINDW as well as of VPORT were chosen as (0.0, 0.0). LT

and WT stand for the length and width respectively in the world coordinate system. XHIGH and YHIGH of the window were chosen according to dimension LT and WT as given below:

For Window, XHIGH = LT

YHIGH = WT

For Viewport,

- XHIGH = 1.0
- YHIGH = WT/LT

} if LT > WT

- XHIGH = LT/WT
- YHIGH = 1.0

} if WT $\geq$ LT

The above criteria was adopted to have larger screen area for picture-display. In case of negative coordinate points, however, one has to choose window parameters in a different fashion keeping in view the mapping of window parameters onto corresponding viewport parameters.

Each of the above procedures was converted into a corresponding program in order to display the corresponding stipple pattern and accordingly the picture was displayed on the graphics terminal.

4.3.1 ProcedureBOXCUT:

This procedure, when called, displays the stipple code (pattern) for nMOS Contact Cut layer in a box (rectangular/square) shape on the display terminal. A check plot (a photographed display) is shown as Plate 4.1. The program is in the file:BOXC.PAS.

4.3.2 Procedure BOXDIFF:

This procedure is for n-MOS Diffusion Layer in a box shape. The photograph of the display is shown as Plate 4.2. The program is in the file:BOXD.PAS.

4.3.3 Procedure BOXIMPLANT:

This is for n-MOS depletion mode implant layer in a box shape.

A photograph of the display is shown as Plate 4.3. The program is in the file:BOXI.PAS.

4.3.4 Procedure BOXMETAL:

This is for n-MOS Metal layer in a box shape. A photograph of the display is shown as Plate 4.4. The file is BOXM.PAS.

4.3.5 Procedure BOXPOLY:

This is for n-MOS polysilicon layer. A photograph of the display is shown as Plate 4.5.The file is BOXP.PAS.

It may be observed that none of the above five patterns is enclosed by continuous solid lines along perimeter of the box. This is a desired feature.

None of the procedures listed above have been described in detail, because the program is fairly readable.

All the procedures after their testing/implementation individually, were placed into the body of the program for interpreter, namely, program INTERPRET, at appropriate places and these procedures will be called as and when needed depending upon the statements/instructions of input CIF file.

4.4 IMPORTANT FEATURES AND LIMITATIONS OF THE INTERPRETER PROGRAM:

a) Important Features:

i) EXJTHINST : As described earlier, this procedure is recursive in nature i.e. this procedure calls itself.

ii) Any number of blanks are allowed at the beginning of instruction and at the end before occurrence of ';'.

iii) It gives error messages.

iv) It caters for inclusion of comments.

v) It caters for nested calls.

b) Limitations:

i) Error reporting is not exhaustive.

iii) Graphics routines for polygons, round flashes and wires have not been designed and implemented. The main reason for not implementing them was that they are not used as commonly as the boxes and secondly, most of the layouts can be drawn using only boxes. These could, however, be implemented in future.

CHAPTER 5

ILLUSTRATIVE EXAMPLES OF SOME SYSTEMS LAYOUTS

Some systems layouts were described in CIF form and photographs (Check plots) of the displayed pictures on the graphics terminal were taken.

5.1 LAYOUT OF SHIFT REGISTER CELL (SRCELL):

SRCELL Layout is described as a collection of boxes on various layers.

5.1.1 Description in CIF Form:

The cell is described in CIF language as below:

```
LNZ;
DS 12 ;
LND;
B 8   8  10   4   1   0;
B 8   8  10   42   1   0;
B 12   18  10   15    1  0;
B 6   4  19   18   1   0;
B 4   2  20   21   1   0;
B 14   4  25   24   1   0;
B 8   8  36   22   1   0;
B 4   14  10   31   1   0;
LNP;
B 20   4  10   12   1   0;
B 4   52  26   26   1   0;
```

```
(CLOCK LINE);
B 10   4   37   12   1   0;
B 8  6    36   17   1   0;
B 12   14   10   29  1    0;
LNC;
B 4  4    10   4  1  0;
B 4  4    10   42  1  0;
B 4  8    36   20  1  0;
B 4  8    10   22  1  0;
LNM;
B 42  8    21   4  1  0;
B 42  8    21   42  1  0;
B 8  12    10   22  1  0;
B 8  12    36   20  1  0;
DF;
C 12 T  0  0;
E
```

5.1.2 Result :

The photograph of the resulting display is shown as Plate 5.1.

(Note: The implant layer has not been drawn so that the other layers may be more easily seen).

5.2 LAYOUT OF A SHIFT REGISTER ARRAY:

The basic constituent of the array is the SRCELL as described in Section 5.1. This SRCELL is defined as a symbol and is then replicated a No. of times in various

layout locations according to translation and mirror parameters in several call statements.

5.2.1 Description in CIF FORM:

The input CIF file is given below:

```
LNZ;
DS 12 ;
LND;
B 8   8   10   4   1   0;
B 8   8   10   42   1   0;
B 12   18   10   15   1  0;
B 6   4   19   18   1   0;
B 4   2   20   21   1   0;
B 14   4   25   24   1   0;
B 8   8   36   22   1   0;
B 4   14   10   31   1   0;
LNP;
B 20   4   10   12   1   0;
B 4   52   26   26   1   0;
(CLOCK LINE);
B 10   4   37   12   1   0;
B 8   6   36   17   1   0;
B 12   14   10   29   1   0;
LNC;
B 4   4   10   4   1   0;
B 4   4   10   42   1   0;
B 4   8   36   20   1   0;
B 4   8   10   22   1   0;
```

```
LNM;
B 42  8   21   4   1   0;
B 42  8   21   42  1   0;
B 8   12  10   22  1   0;
B 8   12  36   20  1   0;
DF;
C 12   T   0   0; C  12  T  0  76;
C 12    T   42  0; C  12  T  42  76;
C  12  T    84   0; C  12  T  84  76;
C 12 T  126  0   ; C  12  T  126  76;
C 12 MY  T    0  84; C  12  MY  T  42  84;
C 12 MY  T   84  84; C  12  MY  T  126  84;
E
```

5.2.2 Result :

Photograph of the resulting display is shown as plate 5.2.

5.3 LAYOUT OF PLAcellpair

We know that PLA is a useful subsystem structure, often used to implement combinational logic and finite-state machines. A PLA can be constructed using six basic cell types (i.e. PLAcellpair, PLAconnect, PLAPulluppair, PLAground, PLAinputdrivers, and PLAoutputinverters and a slight amount of 'random wiring'. Once the layout of these six basic cells is drawn, it is easy to construct CIF descriptions of different sized PLA's having various number of inputs, product terms, and outputs.

5.3.1 Description in CIF Form:

The input CIF file is given below:

```
LNZ;
DS 15;
LND;
B 8   8   4   6   1   0;
B 8   8   4   20   1   0;

(DIFF TO GND);
B 4   28   18   14   1   0;
LNP;
B 4   28   12   14   1   0;
B 4   28   24   14   1   0;
LNC;
B 4   4   4   6   1   0;
B 4   4   4   20   1   0;
LNM;
(METL TO PULL UPS);
B 28   8   14   6   1   0;
B 28   8   14   20   1   0;
DF;
C 15   T   0   0 ;
E
```

5.3.2 Result:

The photograph of the resulting display is shown as Plate 5.3.

The OR and AND planes of the PLA are constructed as arrays of this 14 λ by 14 λ PLACellpair cell. The cell

contains two poly and two metal signal lines and one ground line on the diffusion layer. Diffusion paths may be added at appropriate locations in such cells to form transistors and thus program the PLA.

5.4 LAYOUT OF PLAconnect:

The connection between the AND and OR planes is made by using a PLAconnect. These cells change the signal paths from the metal to the poly layer.

5.4.1 Description in CIF Form:

```
LNZ;
DS 20;
LND;
B 8   8   4   6   1   0;
B 8   8   4   20  1   0;
B 8   8   22  12  1   0;
B 6   4   29  10  1   0;
LNP;
B 20  4   22  4   1   0;
B 20  4   22  20  1   0;
B 6   8   9   6   1   0;
B 6   8   9   20  1   0;
B 4   4   30  16  1   0;
LNC;
B 8   4   6   6   1   0;
B 8   4   6   20  1   0;
B 4   4   22  12  1   0;
```

```
LNM;
B 8   28   22   14   1   0;
B 12   8   6   6   1   0;
B 12   8   6   20   1   0;
DF;
C 20   T   0   0;
E
```

5.4.2 Result:

The photograph of the resulting display is shown as Plate 5.4.

5.5 LAYOUT OF PLA-PULL UP-PAIR:

The pull-up transistors to be placed at the edges of the OR and AND planes are implemented by pull-up-pair cell.

5.5.1 Description in CIF form:

The input CIF file is given below:-

```
LNZ;
DS 25;
LNI;
B 26   10   30   6   1   0;
B 10   16   6   17   1   0;
B 22   10   12   20   1   0;
LND;
B  8   8   4   6   1   0;
B 32   4   24   6   1   0;
B  8   8   6   14   1   0;
B 36   4   22   20   1   0;
B  8   8   22   20   1   0;
```

```
LNP;
B 16   12   28   6   1   0;
B  4   8    38   6   1   0;
B  4   8    18   20  1   0;
B 16   12   8    20  1   0;
B 12   2    6    13  1   0;
LNC;
B  4   4    4    6   1   0;
B  4   4    38   6   1   0;
B  8   4    20   20  1   0;
LNM;
B  8   28   4    14  1   0;
B 12   8    20   20  1   0;
B  6   8    37   6   1   0;
DF;
C 25   T    0    0;
E
```

5.5.2 Result:

The photograph of the resulting display is shown as plate 5.5.

5.6 LAYOUT OF PLAground:

The ground return paths, to be connected to the diffusion lines crossing the planes, are implemented by the PLA-Ground Cell. It is so structured that rows of the cell may be inserted at intervals within AND planes and columns of the cell may be inserted at intervals within OR planes.

5.6.1 Description in CIF Form:

The input CIF file is given below:-

```
LNZ;
DS 30;
LND;
B 4   18   18   11   1   0;
B 8   8   16   10   1   0;
LNP;
B 4   20   8   10   1   0;
B 4   4   12   2   1   0;
B 4   4   12   18   1   0;
B 4   18   24   11   1   0;
LNC;
B 4   4   16   10   1   0;
LNM;
B 28   8   14   10   1   0;
DF;
C 30   T   0   0;
E
```

5.6.2 Result:

The photograph of the resulting display is shown as plate 5.6.

Comments on the results: The clarity of the picture is limited by the device-limitations of Tektronix-display terminal (e.g. one amongst the device limitations is the resolution of 'dots' displayed by the terminal).

CHAPTER 6

USERS' MANUAL

6.1 INTRODUCTION:

This graphics package has been implemented on DEC-1090 system. The program has been written in PASCAL and the graphics routines of the program have been designed using the two basic functions of GPGS. Only 5 of the built-in GPGS routines have been directly called in the program (VPORT, WINDW, NITDEV, LINE (X,Y,0), LINE(X,Y,1)).

6.2 DISPLAY-DEVICE:

Tektronix-4006-1, DVST terminal has been used as the output device.

6.3 SELECTION OF WINDOWS AND VIEWPORT :

The user has to select WINDOW parameters by specifying a rectangular region in the 'user space' (or world coordinate system) as the region of immediate interest. He has to also specify a rectangular portion of the display screen within which his subsequent generated picture is to be displayed called the VIEWPORT. For these, he has to call VPORT and WINDW routines of GPGS by passing the relevant parameters appropriately. User is referred to GPGS Manual (see ref[9]). for details.

If the layout has its XLOW = 0.0 and YLOW = 0.0 in the world coordinate system, he has to assign the values of LT and HT only (LT is XHIGH and HT is YHIGH in the world coordinate), in the program. In such a case, the WINDW and VPORT will be automatically set.

6.4 INSTRUCTION FOR WRITING CIF INPUT FILE:

1. The layout description in CIF form, must be written into the file named INPUT. If it is in some other file, that file should be renamed as INPUT in order to execute the program.

2. To describe the layout in CIF, the user should find out the symbols (i.e. picture segments) which have been replicated at various locations in the overall layout drawn by hand. Existence of any symmetry (which includes translation/replication/mirroring etc.) should be found out to limit the file size.

3. Assign symbol idex, which should be a unique unsigned integer, to each symbol (by 'unique' we mean that the same integer should not be assigned to more than one symbol).

4. All the statements of CIF, except statement E (which represents end of CIF file), must end with ';'.

5. Define all the symbols at the beginning of the file. A symbol must be defined before it is called. To define a symbol proceed as follows:

i) First command is DS command. The format for a DS command will be as follows:-

DS blanks symbol index blanks integer blanks integer [blanks]; A typical DS command will look like:-

DS 100 300 1 .;

(Note: 1.'blanks' in above description means one or more blanks.

2. [blanks] means blanks are optional.

ii) Write the statements pertaining to primitive geometric objects (boxes, only, at the present) in the format given below:-

B blanks integer blanks integer blanks sinteger blanks sinteger blanks sintger blanks sintger [blanks];

Ist integer → length,

2nd integer → width,

First two sintegers specify x,y coordinates of the centre point

7. For 'random wiring', obviously no symbol definition and a call to that symbol is needed.

   The statements for drawing objects pertaining to 'random wiring' (say boxes) are to be written just after the last DF command of symbol definitions.

8. Once the CIF input file has been written down and stored in the file named 'INPUT' proceed as follows to compile/ execute the program:-

   . ASS TTY 3

   . R GPAS

   * File name . PAS

   . EX File name . PAS, SYS: PASLNK.REL,/SEA SYS:GPGS.REL, FORLIB.REL

   (Note: 1. All GPGS routines are recognised by the PASCAL compiler - GPAS.

   2. File name is GICSYS as the program is presently stored in the file GICSYS.PAS).

9. Observe the picture on the terminal. If any errors are there in the picture, edit the INPUT file and again execute the program as given at Sl. 8 to get the desired picture. If needed, photographs can be taken.

10. Notes: 1. For any classifications regarding CIF, the user should refer to ref. [1].

2. For any details about GPGS routines, the user should refer to GPGS Manual [ref. [9])

3. Logical unit No.assigned to TTY should be same as that given in the NITDEV procedure call. (In our case, we have chosen the No. as '3'). Any number between 1 and 7 can be chosen as the logical unit No.

4. GPAS does not check the number or type of parameters passed to GPGS routines and hence extreme care should be taken to see that the correct No. of parameters of appropriate type are provided when GPGS routines are called.

CHAPTER 7

CONCLUSIONS

7.1 TECHNICAL SUMMARY:

GICSYS, a subroutine-based-graphics package has been designed and written in PASCAL and has been implemented on the DEC-1090 system using TEKTRONIX 4006-1, a DVST terminal. The package is meant to be used as a design aid for LSI/VLSI - systems-design. The important graphics routines have been designed and implemented by us while a few of GPGS routines have been called in the program at appropriate places. The routines for stipple-codes (patterns) for different layers of layouts have been exclusively designed and implemented by us (except the fact that it calls only two basic routines of GPGS viz. LINE(X,Y,0) and line (X,Y). The algorithm of these routines are simple. The graphic routines have been linked with the interpreter (core of the project) program which interprets the various commands of CIF file and initiates semantic actions accordingly. The transformation routines have also been exclusively designed and implemented by us.

The parsing table was designed after putting certain constraints and modifying the original syntax. This was done to limit the size of the parsing table which was constructed by hand. Obviously, the table can

be utilized only if the these constraints are adhered to. A more versatile error reporting can be achieved through an automatic parser generator.

7.2 SUGGESTIONS AND SCOPE FOR FUTURE WORK:

1. We have used 'arrays' including 'arrays of records' in our interpreter program. One could think of a data structure which would make the algorithm more efficient. One such possible data structure may be linked list structure in which we can dynamically allocate space as needed and also release unused space.

2. Not much has been done for error handling/recovery in this implementation. A more powerful error handling mechanism can be implemented in future. An automatic parser generator can be implemented for syntactic errors.

3. The graphics routines for polygons,wires and round-flashes on different layers, can be implemented in future.

4. Colour graphics facilities are more suited for the check plots. Colour check plots are much better: colour check plots can be made denser and still be readable; and association of colours with layers and functions is more easily made and subject to fewer errors in practice. These facilities may be created in future.

56. An interactive layout system is more appropriate for design of systems layouts. Here the user can create and modify an integrated system layout directly on the CRT screen. Creating and moving items is fast and easy enough so that the designer can truly sketch on the screen. Once the layout is basically correct, the items can be moved or modified to arrive at the most compact layout. Such a system may be created in future.

APPENDIX A

THE SYNTAX OF CIF-SUBSET IN BNF

[1] <START>::= <CIFFILE> $.

[2] <CIFFILE>::= <COMMAND1LIST><ENDCOMMAND><BLANK> .

[3] <COMMAND1LIST>::=<EMPTY> .

[4] <COMMAND1LIST>::=<COMMAND1LIST><COMMAND1>.

[5] <EMPTY>::= .

[6] <COMMAND1>::=<BLANK><SEMI>.

[7] <COMMAND1>::= <BLANK><COMMAND><SEMI>.

[8] <COMMAND>::= <PRIMCOMMAND>.

[9] <COMMAND>::= <DEFDELETECOMMAND>.

[10] <COMMAND>::= <DEFSTARTCOMMAND><SEMI><PRIMCOMMAND1LIST>
<DEFFINISHOCOMMAND>.

[11] <PRIMCOMMAND1LIST>::= <EMPTY>.

[12] <PRIMCOMMAND1LIST>::= <PRIMCOMMAND1LIST><PRIMCOMMAND1>.

[13] <PRIMCOMMAND1>::= <BLANK><SEMI>.

[14] <PRIMCOMMAND1>::= <BLANK><PRIMCOMMAND><SEMI>.

[15] <PRIMCOMMAND>::= <BOXCOMMAND>.

[16] <PRIMCOMMAND>::= <LAYERCOMMAND>.

[17] <PRIMCOMMAND>::= <CALLCOMMAND>.

[18] <BOXCOMMAND>::=B<INTEGER><SEP><INTEGER><SEP><POINT>
<SEP><POINT>.

[19] <LAYERCOMMAND>::= L<SHORTNAME>.

[20] <DEFSTARTCOMMAND>::= D<EMPTY> S<INTEGER ><SEP>
<INTEGER ><SEP><INTEGER>.

[21] <DEFFINISHCOMMAND>::= D<EMPTY> F.

[22] <DEFDELETECOMMAND>::= D<EMPTY> D<INTEGER>.

[23] <CALLCOMMAND>::= C<INTEGER><TRANSFORMATION>.

[24] <ENDCOMMAND>::= E.

[25] <TRANSFORMATION>::= <TRANSFORMATION1LIST>.

[26] <TRANSFORMATION1LIST>::= <EMPTY>.

[27] <TRANSFORMATION1LIST>::= <TRANSFORMATION1LIST>
<TRANSFORMATION1>.

[28] <TRANSFORMATION1>::= <BLANK> T<POINT>.

[29] <TRANSFORMATION1>::= <BLANK> M<EMPTY> X.

[30] <TRANSFORMATION1>::= <BLANK> M<EMPTY> Y.

[31] <TRANSFORMATION1>::= <BLANK> R<POINT>.

[32] <POINT>::= <SINTEGER><SEP><SINTEGER>.

[33] <SINTEGER>::= <SEP><INTEGERD>.

[34] <SINTEGER>::= <SEP> -<INTEGERD>.

[35] <INTEGER>::= <SEP><INTEGERD>.

[36] <INTEGERD>::= <DIGIT><DIGITLIST>.

[37] <DIGITLIST>::= <EMPTY>.

[38] <DIGITLIST>::= <DIGITLIST><DIGIT>.

[39] <SHORTNAME>::= <C1><C1>.

[40] <C1>::= <UPPERCHAR>.

[41] <SEMI>::= ; .

[42] < SEP>::= <BLANK>.

[43] <DIGIT>::= 0.

[44] <DIGIT>::= 1.

[45] <DIGIT>::= 2.

[46] <DIGIT>::= 3.

[47] <DIGIT>::= 4.

[48] <DIGIT>::= 5.

[49] <DIGIT>::= 6.

[50] <DIGIT>::= 7.

[51] <DIGIT>::= 8.

[52] <DIGIT>::= 9.

[53] <UPPERCHAR>::= A.

[54] <UPPERCHAR>::= B.

[55] <UPPERCHAR>::= C.

[56] <UPPERCHAR>::= D

[57] <UPPERCHAR>::= E.

[58] <UPPERCHAR>::= F.

[59]: <UPPERCHAR>::= G.

[60] <UPPERCHAR>::= H.

[61] <UPPERCHAR>::= I.

[62] <UPPERCHAR>::= J.

[63] <UPPERCHAR>::= K.

[64] <UPPERCHAR>::= L.

[65] <UPPERCHAR>::= M.

[66] <UPPERCHAR>::= N.

[67] <UPPERCHAR>::= O.

[68] <UPPERCHAR>::= P.

[69] <UPPERCHAR>::= Q.

[70] <UPPERCHAR>::= R.

[71] <UPPERCHAR>::= S.

[72] <UPPERCHAR>::= T.

[73] <UPPERCHAR>::= U.

[74] <UPPERCHAR>::= V.

[75] <UPPERCHAR>::= W.

[76] <UPPERCHAR>::= X.

[77] <UPPERCHAR>::= Y.

[78] <UPPERCHAR>::= Z.

[79] <BLANK>::= BLANK.

APPENDIX B

SLR PARSING TABLE

Note 1: All the symbols i.e. terminals and non-terminals have been indexed as follows:

TERMINAL-DETAILS

TNO	TERMINAL NAME
0	e
1	$
2	B
3	L
4	D
5	S
6	F
7	C
8	E
9	T
10	M
11	X
12	Y
13	R
14	-
15	;
16	0
17	1
18	2
19	3
20	4
21	5

22 = 6
23 = 7
24 = 8
25 = 9
26 = A
27 = G
28 = H
29 = I
30 = J
31 = K
32 = N
33 = O
34 = P
35 = Q
36 = U
37 = V
38 = W
39 = Z
40 = .

NONTERMINAL-DETAILS

NONT	NONTERMINAL NAME
41	START
42	CIFFILE
43	COMMAND1LIST
44	ENDCOMMAND
45	BLANK
46	EMPTY
47	COMMAND1
48	SEMI
49	COMMAND
50	PRIMCOMMAND

51 = DEFDELETECOMAND
52 = DEFSTARTCOMMAND
53 = PRIMCOMMAND1LIST
54 = DEFFINISHCOMMAND
55 = PRIMCOMMAND1
56 = BOXCOMMAND
57 = LAYERCOMMAND
58 = CALLCOMMAND
59 = INTEGER
60 = SEP
61 = POINT
62 = SHORTNAME
63 = TRANSFORMATION
64 = TRANSFORMATION1LIST
65 = TRANSFORMATION1
66 = SINTEGER
67 = INTEGERD
68 = DIGIT
69 = DIGITLIST
70 = C1
71 = UPPERCHAR

Note 2: The entries of the table have been tabulated row-wise. The row No. (i.e. the state number) has been enclosed within * *.

Note 3: The first argument of the ACTION function is the row No. (i.e. state No.) while its 2nd argument indicates the column No. which is the index for some grammar symbol. As a typical example,

ACTION (1,D) = r5 has been represented in this table as:- * 1 * [4] : r5 where 4 is the col.No. which is the index for D.

Note 4: The 1st argument of the GOTO function is the row No. (i.e. state No.) while its 2nd argument is the col. No. As a typical example, GOTO (3, endCommand) = 7 has been represented in the table as:- * 3 * [44]:PG7 where 44 is the colmn No. which is the index for endCommand.

Note 5: All the missing entries (i.e. those entries which have not been reflected in this table) are 'error' entries.

0 [0] : s1, [42]:PG2, [43]:PG3, [46]:PG4

1 [4]: r5, [5]:r5, [6]:r5, [8]:r5,[11]:r5, [12]:r5, [15]:r5, [16]:r5, [17]:r5, [18]:r5, [19]:r5, [20]:r5, [21]:r5, [22]:r5, [23]:r5, [24]:r5, [25]:r5, [40]:r5

2 [1]: ACC

3 [8]:s5, [40]:s6, [44]:PG7, [45]:PG8, [47]:PG9

4 [8]:r3, [40]:r3

5 [40]:r24

6 [1]:r79, [2]:r79, [3]:r79,,[4]:r79, [7]:r79, [9]:r79, [10]:r79, [13]:r79, [14]:r79, [15]:r79, [16]:r79, [17]:r79, [18]:r79, [19]:r79, [20]:r79, [21]:r79, [22]:r79, [23]:r79, [24]:r79, [25]:r79, [40]:r79

7 [40]:s6, [45]:PG10

8 [2]:s11, [3]:s12, [4]:s13, [7]:s14, [15]:s15,[48]:PG16, [49]:PG17, [50]:PG18, [51]:PG19, [52]:PG20, [56]:PG21, [57]:PG22, [58]:PG23

9 [8]:r4, [40]:r4

10 [1]:r2

11 [40]:s6, [45]:PG24, [59]:PG25, [60]:PG26

12 [2]:s27, [3]:s28, [4]:s29, [5]:s30, [6]:s31, [7]:s32, [8]:s33, [9]:s34, [10]:s35, [11]:s36, [12]:s37, [13]:s38, [28]:s39, [27]:s40, [28]:s41, [29]:s42, [30]:s43, [31]:s44, [32]:s45, [33]:s46, [34]:s47, [35]:s48, [36]:s49, [37]:s50, [38]:s51, [39]:s52, [62]:PG53, [70]: PG54, [71]: PG55

13 [0]: s1, [46]:PG56

14 [40]:s6, [45]:PG24, [59]:PG57, [60]:PG26

15 [4]:r41, [8]:r41,[40]:r41

16 [8]:r6, [40]:r6

17 [15]:s15, [48]:PG58

18 [15]:r8

19 [15]:r9,

20 [15]:s15, [48]:PG59

21 [15]:r15

22 [15]:r16

23 [15]:r17

24 [14]:r42, [16]:r42, [17]:r42, [18]:r42, [19]:r42, [20]:r42, [21]:r42, [22]:r42, [23]:r42, [24]:r42, [25]:r42, [40]:r42

25 [40]:s6, [45]:PG24, [60]:PG60

26 [16]:s61, [17]:s62, [18]:s63, [19]:s64, [20]:s65, [21]:s66, [22]:s67, [23]:s68, [24]:s69, [25]:s70, [67]:PG71, [68]:PG72

27 [2]:r54, [3]:r54, [4]:r54, [5]:r54, [6]:r54, [7]:r54, [8]:r54, [9]:r54, [10]:r54, [11]:r54, [12]:r54, [13]:r54, [15]:r54, [26]:r54, [27]:r54, [28]:r54, [29]:r54, [30]:r54, [31]:r54, [32]:r54, [33]:r54, [34]:r54, [35]:r54, [36]:r54, [37]:r54, [38]:r54, [39]:r54

28 [2]:r64, [3]:r64, [4]:r64, [5]:r64, [6]:r64, [7]:r64, [8]:r64, [9]:r64, [10]:r64, [11]:r64, [12]:r64, [13]:r64, [15]:r64, [26]:r64, [27]:r64, [28]:r64, [29]:r64, [30]:r64, [31]:r64, [32]:r64, [33]:r64, [34]:r64, [35]:r64, [36]:r64, [37]:r64, [38]:r64, [39]:r64

29 [2]:r56, [3]:r56, [4]:r56, [5]:r56, [6]:r56, [7]:r56, [8]:r56, [9]:r56, [10]:r56, [11]:r56, [12]:r56, [13]:r56, [15]:r56, [26]:r56, [27]:r56, [28]:r56, [29]:r56, [30]:r56, [31]:r56, [32]:r56, [33]:r56, [34]:r56, [35]:r56, [36]:r56, [37]:r56, [38]:r56, [39]:r56

30 [2]:r71, [3]:r71, [4]:r71, [5]:r71, [6]:r71, [7]:r71, [8]:r71, [9]:r71, [10]:r71, [11]:r71, [12]:r71, [13]:r71, [15]:r71,[26]:r71, [27]:r71, [28]:r71, [29]:r71, [30]:r71, [31]:r71, [32]:r71, [33]:r71, [34]:r71, [35]:r71,[36]:r71, [38]:r71, [39]:r71

31 [2]:r58, [3]:r58, [4]:r58, [5]:r58, [6]:r58, [7]:r58, [8]:r58, [9]:r58, [10]:r58, [11]:r58, [12]:r58, [13]:r58, [15]:r58, [26]:r58, [27]:r58, [28]:r58, [29]:r58,[30]:r58, [31]:r58, [32]:r58, [33]:r58, [34]:r58, [35]:r58,[36]:r58, [37]:r58, [38]:r58,[39]:r58

32* [2]:r55, [3]:r55, [4]:r55, [5]:r55, [6]:r55, [7]:r55, [8]:r55, [9]:r55, [10]:r55, [11]:r55, [12]:r55, [13]:r55, [15]:r55, [26]:r55, [27]:r55, [28]:r55, [29]:r55, [30]:r55, [31]:r55, [32]:r55, [33]:r55, [34]:r55, [35]:r55, [36]:r55, [37]:r55, [38]:r55, [39]:r55.

33* [2]:r57, [3]:r57, [4]:r57, [5]:r57, [6]:r57, [7]:r57, [8]:r57, [9]:r57, [10]:r57, [11]:r57, [12]:r57, [13]:r57, [15]:r57, [26]:r57, [27]:r57, [28]:r57, [29]:r57, [30]:r57 [31]:r57, [32]:r57, [33]:r57, [34]:r57, [35]:r57, [36]:r57 [37]:r57, [38]:r57, [39]:r57

34* [2]:r72, [3]:r72, [4]:r72, [5]:r72, [6]:r72, [7]:r72, [8]:r72, [9]:r72, [10]:r72, [11]:r72, [12]:r72,[13]:r72,

[15]:r72, [26]:r72, [27]:r72, [28]:r72, [29]:r72,[30]:r72,
[31]:r72, [32]:r72, [33]:r72, [34]:r72, [35]:r72,[36]:r72,
[37]:r72, [38]:r72, [39]:r72

35 [2]:r65, [3]:r65, [4]:r65, [5]:r65, [6]:r65, [7]:r65,
[8]:r65, [9]:r65, [10]:r65, [11]:r65, [12]:r65, [13]:r65,
[15]:r65,[26]:r65, [27]:r65, [28]:r65, [29]:r65, [30]:r65
[31]:r65, [32]:r65, [33]:r65, [34]:r65, [35]:r65, [36]:r65,
[37]:r65, [38]:r65, [39]:r65

36 [2]:r76, [3]:r76, [4]:r76, [5]:r76, [6]:r76, [7]:r76,
[8]:r76, [9]:r76, [10]:r76, [11]:r76, [12]:r76, [13]:r76,
[15]:r76, [26]:r76, [27]:r76, [28]:r76, [29]:r76, [30]:r76,
[31]:r76, [32]:r76, [33]:r76, [34]:r76, [35]:r76, [36]:r76,
[37]:r76, [38]:r76, [39]:r76

37 [2]:r77, [3]:r77, [4]:r77, [5]:r77, [6]:r77, [7]:r77,
[8]:r77, [9]:r77, [10]:r77, [11]:r77, [12]:r77, [13]: r77,
[15]:r77, [26]:r77, [27]:r77, [28]:r77, [29]:r77, [30]:r77,
[31]:r77, [32]:r77, [33]:r77, [34]:r77, [35]:r77, [36]:r77,
[37]:r77, [38]:r77, [39]:r77.

38 [2]:r70, [3]:r70, [4]:r70, [5]:r70, [6]:r70, [7]:r70,
[8]:r70, [9]:r70, [10]:r70, [11]:r70, [12]:r70,[13]:r70,
[15]:r70, [26]:r70, [27]:r70, [28]:r70, [29]:r70, [30]:r70,
[31]:r70, [32]:r70, [33]:r70, [34]:r70, [35]:r70, [36]:r70,
[37]:r70, [38]:r70, [39]:r70.

9* [2]:r53, [3]:r53, [4]:r53, [5]:r53 [6]:r53, [7]:r53,
[8]:r53, [9]:r53, [10]:r53, [11]:r53, [12]:r53, [13]:r53,
[15]:r53, [26]:r53, [27]:r53, [28]:r53, [29]:r53, [30]:r53,
[31]:r53, [32]:r53, [33]:r53, [34]:r53, [35]:r53, [36]:r53,
[37]:r53, [38]:r53, [39]:r53.

0* [2]:r59, [3]:r59, [4]:r59, [5]:r59, [6]:r59, [7]:r59,
[8]:r59, [9]:r59, [10]:r59, [11]:r59, [12]:r59, [13]:r59,
[15]:r59, [26]:r59, [27]:r59, [28]:r59, [29]:r59, [30]:r59,
[31]:r59, [32]:r59, [33]:r59, [34]:r59, [35]:r59, [36]:r59,
[37]:r59, [38]:r59, [39]:r59

* [2]:r60, [3]:r60, [4]:r60, [5]:r60, [6]:r60, [7]:r60,
[8]:r60, [9]:r60, [10]:r60, [11]:r60, [12]:r60, [13]:r60,
[15]:r60, [26]:r60, [27]:r60, [28]:r60, [29]:r60, [30]:r60,
[31]:r60, [32]:r60, [33]:r60, [34]:r60, [35]:r60, [36]:r60,
[37]:r60, [38]:r60, [39]:r60

* [2]:r61, [3]:r61, [4]:r61, [5]:r61, [6]:r61, [7]:r61,
[8]:r61, [9]:r61, [10]:r61, [11]:r61, [12]:r61, [13]:r61,
[15]:r61, [26]:r61, [27]:r61, [28]:r61, [29]:r61, [30]:r61,
[31]:r61, [32]:r61, [33]:r61, [34]:r61, [35]:r61, [36]:r61,
[37]:r61, [38]:r61, [39]:r61.

* [2]:r62, [3]:r62, [4]:r62, [5]:r62, [6]:r62, [7]:r62,
[8]:r62, [9]:r62, [10]:r62, [11]:r62, [12]:r62, [13]:r62,
[15]:r62, [26]:r62, [27]:r62, [28]:r62, [29]:r62, [30]:r62,
[31]:r62, [32]:r62, [33]:r62, [34]:r62, [35]:r62,
[36]:r62, [37]:r62,[38]:r62, [39]:r62

4* [2]:r63, [3]:r63, [4]:r63, [5]:r63, [6]:r63, [7]:r63,
[8]:r63, [9]:r63, [10]:r63, [11]:r63, [12]:r63, [13]:r63,
[15]:r63, [26]:r63, [27]:r63, [28]:r63, [29]:r63, [30]:r63,
[31]:r63, [32]:r63, [33]:r63, [34]:r63, [35]:r63, [36]:r63,
[37]:r63, [38]:r63, [39]:r63

5* [2]:r66, [3]:r66, [4]:r66, [5]:r66, [6]:r66, [7]:r66,
[8]:r66, [9]:r66, [10]:r66, [11]:r66, [12]:r66, [13]:r66,
[15]:r66, [26]:r66, [27]:r66, [28]:r66, [29]:r66, [30]:r66,
[31]:r66, [32]:r66, [33]:r66, [34]:r66, [35]:r66, [36]:r66,
[37]:r66, [38]:r66, [39]:r66

6* [2]:r67, [3]:r67, [4]:r67, [5]:r67, [6]:r67, [7]:r67,
[8]:r67, [9]:r67, [10]:r67, [11]:r67, [12]:r67, [13]:r67,
[15]:r67, [26]:r67, [27]:r67, [28]:r67, [29]:r67, [30]:r67,
[31]:r67, [32]:r67, [33]:r67, [34]:r67, [35]:r67, [36]:r67
[37]:r67, [38]:r67, [39]:r67

7* [2]:r68, [3]:r68, [4]:r68, [5]:r68, [6]:r68, [7]:r68,
[8]:r68, [9]:r68, [10]:r68, [11]:r68, [12]:r68, [13]:r68,
[15]:r68, [26]:r68, [27]:r68, [28]:r68, [29]:r68, [30]:r68,
[31]:r68, [32]:r68, [33]:r68, [34]:r68, [35]:r68, [36]:r68,
[37]:r68, [38]:r68, [39]:r68

8* [2]:r69, [3]:r69, [4]:r69, [5]:r69, [6]:r69, [7]:r69,
[8]:r69, [9]:r69, [10]:r69, [11]:r69, [12]:r69, [13]:r69,

44 [2]:r63, [3]:r63, [4]:r63, [5]:r63, [6]:r63, [7]:r63, [8]:r63, [9]:r63, [10]:r63, [11]:r63, [12]:r63, [13]:r63, [15]:r63, [26]:r63, [27]:r63, [28]:r63, [29]:r63, [30]:r63, [31]:r63, [32]:r63, [33]:r63, [34]:r63, [35]:r63, [36]:r63, [37]:r63, [38]:r63, [39]:r63

45 [2]:r66, [3]:r66, [4]:r66, [5]:r66, [6]:r66, [7]:r66, [8]:r66, [9]:r66, [10]:r66, [11]:r66, [12]:r66, [13]:r66, [15]:r66, [26]:r66, [27]:r66, [28]:r66, [29]:r66, [30]:r66, [31]:r66, [32]:r66, [33]:r66, [34]:r66, [35]:r66, [36]:r66, [37]:r66, [38]:r66, [39]:r66

46 [2]:r67, [3]:r67, [4]:r67, [5]:r67, [6]:r67, [7]:r67, [8]:r67, [9]:r67, [10]:r67, [11]:r67, [12]:r67, [13]:r67, [15]:r67, [26]:r67, [27]:r67, [28]:r67, [29]:r67, [30]:r67, [31]:r67, [32]:r67, [33]:r67, [34]:r67, [35]:r67, [36]:r67, [37]:r67, [38]:r67, [39]:r67

47 [2]:r68, [3]:r68, [4]:r68, [5]:r68, [6]:r68, [7]:r68, [8]:r68, [9]:r68, [10]:r68, [11]:r68, [12]:r68, [13]:r68, [15]:r68, [26]:r68, [27]:r68, [28]:r68, [29]:r68, [30]:r68, [31]:r68, [32]:r68, [33]:r68, [34]:r68, [35]:r68, [36]:r68, [37]:r68, [38]:r68, [39]:r68

48 [2]:r69, [3]:r69, [4]:r69, [5]:r69, [6]:r69, [7]:r69, [8]:r69, [9]:r69, [10]:r69, [11]:r69, [12]:r69, [13]:r69,

[15]:r69, [26]:r69, [27]:r69, [28]:r69, [29]:r69, [30]:r69, [31]:r69, [32]:r69, [33]:r69, [34]:r69, [35]:r69, [36]:r69, [37]:r69, [38]:r69, [39]:r69

49 [2]:r73, [3]:r73, [4]:r73, [5]:r73, [6]:r73, [7]:r73, [8]:r73, [9]:r73, [10]:r73, [11]:r73, [12]:r73, [13]:r73, [15]:r73, [26]:r73, [27]:r73, [28]:r73, [29]:r73, [30]:r73, [31]:r73, [32]:r73, [33]:r73, [34]:r73, [35]:r73, [36]:r73, [37]:r73, [38]:r73, [39]:r73

50 [2]:r74, [3]:r74, [4]:r74, [5]:r74, [6]:r74, [7]:r74, [8]:r84, [9]:r74, [10]:r74, [11]:r74, [12]:r74, [13]:r74, [15]:r74, [26]:r74, [27]:r74, [28]:r74, [29]:r74, [30]:r74, [31]:r74, [32]:r74, [33]:r74, [34]:r74, [35]:r74, [36]:r74, [37]:r74, [38]:r74, [39]:r74

51 [2]:r75, [3]:r75, [4]:r75, [5]:r75, [6]:r75, [7]:r75, [8]:r75, [9]:r75, [10]:r75, [11]:r75, [12]:r75, [13]:r75, [15]:r75, [26]:r75, [27]:r75, [28]:r75, [29]:r75, [30]:r75, [31]:r75, [32]:r75, [33]:r75, [34]:r75, [35]:r75, [36]:r75, [37]:r75, [38]:r75, [39]:r75

52 [2]:r78, [3]:r78, [4]:r78, [5]:r78, [6]:r78, [7]:r78, [8]:r78, [9]:r78, [10]:r78, [11]:r78, [12]:r78, [13]:r78, [15]:r78, [26]:r78, [27]:r78, [28]:r78, [29]:r78, [30]:r78, [31]:r78, [32]:r78, [33]:r78, [34]:r78, [35]:r78, [36]:r78, [37]:r78, [38]:r78, [39]:r78.

53 [15]:r19

54 [2]:s27, [3]:s28, [4]:s29, [5]:s30, [6]:s31, [7]:s32,
[8]:s33, [9]:s34, [10]:s35, [11]:s36, [12]:s37, [13]:s38,
[26]:s39, [27]:s40, [28]:s41, [29]:s42, [30]:s43, [31]:s44,
[32]:s45, [33]:s46, [34]:s47, [35]:s48, [36]:s49, [37]:s50,
[38]:s51, [39]:s52, [70]:PG73, [71]:PG55

55 [2]:r40, [3]:r40, [4]:r40, [5]:r40, [6]:r40, [7]:r40,
[8]:r40, [9]:r40, [10]:r40, [11]:r40, [12]:r40, [13]:r40,
[15]:r40, [26]:r40, [27]:r40, [28]:r40, [29]:r40, [30]:r40,
[31]:r40, [32]:r40, [33]:r40, [34]:r40, [35]:r40, [36]:r40,
[37]:r40, [38]:r40, [39]:r40

56 [4]:s74, [5]:s75.

57 [0]:s1, [46]:PG76, [63]:PG77, [64]:PG78.

58 [8]:r7, [40]:r7.

59 [0]:s1, [46]:PG79, [53]:PG80

60 [40]:s6, [45]:PG24, [59]:PG81, [60]:PG26.

61 [15]:r43, [16]:r43, [17]:r43, [18]:r43, [19]:r43, [20]:r43,
[21]:r43, [22]:r43,, [23]:r43, [24]:r43, [25]:r43, [40]:r43

62 [15]:r44, [16]:r44, [17]:r44, [18]:r44, [19]:r44, [20]:r44,
[21]:r44, [22]:r44, [23]:r44, [24]:r44, [25]:r44, [40]:r44

63 [15]:r45, [16]:r45, [17]:r45, [18]:r45, [19]:r45, [20]:r45,
[21]:r45, [22]:r45, [23]:r45, [24]:r45, [25]:r45, [40]:r45

64 [15]:r46, [16]:r46, [17]:r46, [18]:r46, [19]:r46, [20]:r46,
[21]:r46, [22]:r46, [23]:r46, [24]:r46, [25]:r46, [40]:r46

65 [15]:r47, [16]:r47, [17]:r47, [18]:r46, [19]:r47, [20]:r47,
[21]:r47, [22]:r47, [23]:r47, [24]:r47, [25]:r47, [40]:r47

66 [15]:r48, [16]:r48, [17]:r48, [18]:r48, [19]:r48, [20]:r48,
[21]:r48, [22]:r48, [23]:r48, [24]:r48, [25]:r48, [40]:r48

67 [15]:r49, [16]:r49, [17]:r49, [18]:r49, [19]:r49, [20]:r49,
[21]:r49, [22]:r49, [23]:r49, [24]:r49, [25]:r49, [40]:r49

68 [15]:r50, [16]:r50, [17]:r50, [18]:r50, [19]:r50, [20]:r50,
[21]:r50, [22]:r50, [23]:r50, [24]:r50, [25]:r50, [40]:r50

69 [15]:r51,[16]:r51, [17]:r51, [18]:r51, [19]:r51, [20]:r51,
[21]:r51, [22]:r51, [23]:r51, [24]:r51, [25]:r51, [40]:r51

70 [15]:r52, [16]:r52, [17]:r52, [18]:r52, [19]:r52, [20]:r52,
[21]:r52, [22]:r52, [23]:r52, [24]:r52, [25]:r52, [40]:r52

71 [15]:r35, [40]:r35

72 [0]:s1, [46]:PG82, [69]:PG83.

73 [15]:r39

74 [40]:s6, [45]:PG24, [59]:PG84, [60]:PG26

75 [40]:s6, [45]:PG24, [59]:PG84, [60]:PG26

76 [15]:r26, [40]:r26

77 [15]:r23

78 [15]:r25, [40]:s6, [45]:PG86, [65]:PG87.

79 [4]:r11, [40]:r11

80 [4]:s88, [40]:s6, [45]:PG89, [54]:PG90, [55]:PG91

81 [40]:s6, [45]:PG24, [60]:PG92

82 [15]:r37, [16]:r37, [17]:r37, [18]:r37, [19]:r37, [20]:r37, [21]:r37, [22]:r37, [23]:r37, [24]:r37, [25]:r37, [40]:r37

83 [15]:r36, [16]:s61, [17]:s62, [18]:s63, [19]:s64, [20]:s65, [21]:s66, [22]:s67, [23]:s68, [24]:s69, [25]:s70, [40]:r36, [68]:PG93

84 [15]:r22.

85 [40]:s6, [45]:PG24, [60]:PG94

86 [9]:s95, [10]:s96, [13]:s97

87 [15]:r27, [40]:r27

88 [0]:s1, [46]:PG98.

89 [2]:s11, [3]:s12, [7]:s14, [15]:s15, [48]:PG99, [50]:PG100, [56]:PG21, [57]:PG22, [58]:PG23.

90 [15]:r10.

91 [4]:r12, [40]:r12

92 [40]:s6, [45]:PG24, [60]:PG101, [61]:PG102, [66]:PG103

93 [15]:r38, [16]:r38, [17]:r38, [18]:r38, [19]:r38, [20]:r38, [21]:r38, [22]:r38, [23]:r38, [24]:r38, [25]:r38, [40]:r38

94 [40]:s6, [45]:PG24, [59]:PG104, [60]:PG26

95 [40]:s6, [45]:PG24, [60]:PG101, [61]:PG105, [66]:PG103

96 [0]:s1, [46]:PG106

97 [40]:s6, [45]:PG24, [60]:PG101, [61]:PG107, [66]:PG103

98 [6]:s108

99 [4]:r13, [40]:r13

100 [15]:s15, [48]:PG109

101 [14]:s110, [16]:s61, [17]:s62, [18]:s63, [19]:s64, [20]:s65, [21]:s66, [22]:s67, [23]:s68, [24]:s69, [25]:s70, [67]:PG111, [68]:PG72

102 [40]:s6, [45]:PG24, [60]:PG112

103 [40]:s6, [45]:PG24, [60]:PG113

104 [40]:s6, [45]:PG24, [60]:PG114

105 [15]:r28, [40]:r28

106 [11]:s115, [12]:s116,

107 [15]:r31, [40]:r31

108 [15]:r21

109 [4]:r14, [40]:r14.

110 [16]:s61, [17]:s62, [18]:s63, [19]:s64, [20]:s65, [21]:s66, [22]:s67, [23]:s68, [24]:s69, [25]:s70, [67]:PG117, [68]:PG72

111 [15]:r33, [40]:r33

112 [40]:s6, [45]:PG24, [60]:PG101, [61]:PG118, [66]:PG103

113 [40]:s6, [45]:PG24, [60]:PG101, [66]:PG119

114 [40]:s6, [45]:PG24, [59]:PG120, [60]:PG26

115 [15]:r29, [40]:r29

116 [15]:r30, [40]:r30

117 [15]:r34, [40]:r34

118 [15]:r18

119 [15]:r32, [40]:r32

120 [15]:r20.

REFERENCES

1. Mead, Carver and Conway, Lynn: 'Introduction to VLSI Systems,' Addision-Wesley Publishing Company, 1980.

2. N. Wirth, 'What can we do about the unnecessary diversity of Notations for syntactic definitions?', Communication of the ACM, November 1977.

3. Aho, A.V., and Ulman, J.D.:' Principles of Compiler Design,' Addison-Wesley Publishing Company, 1977.

4. Hunter, Robin:'The design and construction of compilers.' John Wiley and Sons, 1983.

5. DHAMDHERE, D.M. 'Compiler Construction- Principles and Practice', MacMillan India Ltd., 1983.

6. Newman, M., Sproull, F., 'Principles of Interactive Computer Graphics', 2nd edition, McGraw Hill, Kogakusha, 1979.

7. Foley, J.D., VANDAM, A., 'Fundamentals of Interactive Computer Graphics', Addison Wesley Publishing Co. 1982.

8. Steven Harrington, 'Computer Graphics - a programming approach', McGraw Hill, 1983.

9. MUDUR, S.P., 'General Purpose Graphics System User's Manual', NCSDCT, TIFR, Bombay 400005, India.

10. Rajaraman, V.: 'Computer Programming in Pascal', Prentice Hall of India Pvt. Ltd., 1983.

11. Mavor, J: 'Introduction to MOS LSI DESIGN', Addison - Wesley Publishing Co., 1983.

12. Ayres, R.F., 'Silicon Compilation and the Art of Automatic Microchip Design' Prentice-Hall Inc. New Jersey, 1983.

EE-1985-M-AHM-DES